

QW- Editor

Reference Guide

*QuickWave is a general purpose electromagnetic simulator
based on conformal finite-difference time-domain method.*

version 6.5

Copyright QWED

Warsaw 2007

E 1. FUNDAMENTALS OF THE METHOD OF ANALYSIS	5
E 2. GRAPHICAL EDITOR	7
E 2.1. Main menu and submenus of QW-Editor	7
E 2.1.1. <i>File</i> group of commands	7
E 2.1.2. <i>Parameters</i> group of commands	8
E 2.1.2.1. AMIGO	9
E 2.1.3. <i>Windows</i> group of commands	12
E 2.1.4. <i>Project</i> group of commands	13
E 2.1.5. <i>Tools</i> group of commands	13
E 2.1.6. <i>Help</i> group of commands	16
E 2.1.7. Overview of icons within QW-Editor main window	17
E 2.1.8. Displays of the shape of QW-3D projects	17
E 2.1.9. Displays of the shape of QW-V2D projects	19
E 2.2. The 2D windows	20
E 2.2.1. The <i>Setup</i> group of commands	21
E 2.2.2. The <i>Edit</i> group of commands	27
E 2.2.3. The <i>Draw</i> group of commands	34
E 2.2.4. The <i>View</i> group of commands	37
E 2.2.5. The <i>Info</i> and <i>Help</i> groups of commands	38
E 2.2.6. Toolbar of 2D window	38
E 2.2.7. Additional remarks on wire grids (asymptotic boundary conditions)	39
E 2.3. The 3D windows	42
E 2.4. Drawing and editing ports	44
E 2.4.1. Drawing I/O Ports and reference planes	44
E 2.4.2. Drawing mesh snapping planes and symmetry planes	49
E 2.4.3. Editing ports	50
E 2.5. Parameters of wave simulation	50
E 2.5.1. Input/Output Port parameters	50
E 2.5.1.1. I/O Port Parameters in the case of transmission line ports	51
E 2.5.1.2. I/O Port parameters in the case of absorbing walls	55
E 2.5.1.3. I/O Port parameters in the case of lumped sources/probes or lumped impedances	56
E 2.5.1.4. I/O Port parameters in the case of free space excitation by a plane wave or a Gaussian beam	57
E 2.5.1.5. I/O Port parameters in the case of QW-V2D ports	58
E 2.5.2. Processing/Postprocessing	58
E 2.5.3. Media parameters	63
E 2.5.4. Units	69
E 2.6. Application of QW-Editor using library objects	69

E 2.7. File exporting and starting the simulation from <i>QW-Editor</i>	70
E 2.7.1. Export and Run commands	70
E 2.7.2. How to assure the proper mesh generation	70
E 2.8. Command line options of <i>QW-Editor</i>	72
E 3. SYNTAX OF THE USER DEFINED OBJECT (UDO) LANGUAGE	73
E 3.1. What is the User Defined Object	73
E 3.2. Structure of a single UDO and calls to other UDO's	73
E 3.3. Defining elements, combined elements and materials	76
E 3.4. Variables, functions, arithmetic expressions and file reading	80
E 3.5. Loops in UDO language	83
E 3.6. Defining ports, boxes, mesh snapping planes, postprocessing and circuit parameters	84
E 3.7. Moving the elements, copying them and controlling their intersections	90
E 3.8. Biphased element and objects (<i>QW-3D</i> only)	95
E 3.9. Controlling the mesh	95
E 3.10. Files used by UDO language and their location	96
E 3.11. Optimising with UDO	98
E 3.12. UDO commands and functions in alphabetic order	99
E 4. FILES CREATED OR USED BY <i>QW-EDITOR</i>	104
E 4.1. Files used by <i>QW-Editor</i>, created by <i>QW-Editor</i> or externally	104
E 4.2. Files created by <i>QW-Editor</i> and used by <i>QW-Simulator</i>	105
E 4.2.1. Basic structure of shape files	106
E 5. INDEX	107

E 1. Fundamentals of the method of analysis

The electromagnetic fields are in general characterised by three-dimensional Maxwell equations. In the Finite Difference Time Domain method (FDTD) these equations are directly discretised (in space and time) and solved explicitly. The discretisation in space means that the considered domain should be divided into small subdomains called cells. Since the time variable is also discretised we calculate the field distribution in consecutive moments separated by the time step Δt . The algorithm simulates the natural processes in the considered space. The increments of the E -field components between the time instants t_n and $t_{n+1} = t_n + \Delta t$ are calculated from the space derivatives of the H -components at the instant $t_n + 0.5 \Delta t$. Next, the increments of the H -field components between the time instants $t_{n+0.5}$ and $t_{n+1.5} = t_{n+0.5} + \Delta t$ are calculated from the space derivatives of the E -components at the instant t_{n+1} . The process of field calculation for a consecutive time instant advanced by Δt is called iteration.

FDTD was first introduced by Yee in 1966 and since then applied by many authors. It is currently one of the most popular methods among microwave circuit researchers. The version of the FDTD method applied here is based on the research conducted by W. K. Gwarek and his co-workers. The main difference with respect to the classical FDTD method is much more flexibility in the shape of individual cells. This reduces the main disadvantage of the classical FDTD, which is the necessity of stair case approximation of curved boundaries. *QW-3D* software also contains many other features based on the original research (see [Chapter UG 3](#) or [V2D 3](#) for references).

In most of the FDTD applications we use a pulse excitation at one or more ports of the modelled device. After the wave simulation is accomplished, the comparison of the Fourier transforms of the input and output signals gives the S-matrix parameters of the circuit. Furthermore, the Fourier transforms of tangential fields on a surface enclosing the antenna or a scatterer can be used to extract the radiated or scattered far fields, respectively. Theoretically, the Fourier transform calculations require infinite period of time. However, since the exciting pulse has limited duration and the power entering the circuit is being dissipated at the input and output (which are matched), or at the absorbing boundaries, the signal at the ports becomes negligible after a limited period of time and the Fourier transform calculation can be limited to this period without causing significant errors.

As all numerical methods the FDTD method has its limits of accuracy for given computer resources. There are two basic sources of errors:

a) Error due to finite discretisation of space (finite cell size)

The division of the circuit into cells (called meshing) is performed in the *QW-Editor* automatically but it is controlled by the user. The user chooses the basic cell size, which determines the accuracy on one hand, and the computer time and memory needed for the simulations on the other hand. The best way to check the level of the discretisation error for a particular application is to analyse the same circuit with different cell size and to compare the results. In *QW-3D* reducing the cell size by half brings the space discretisation errors down by a factor of 4, or nearly 4. However, it should also be mentioned that the computing time will rise almost 16 times and memory occupation 8 times.

b) Error due to finite computing time used for wave simulation and the stop criteria for FDTD simulations

This error results from the fact that calculation of Fourier transforms of the input and output signals is restricted to the number of iterations, in which not all the energy injected into the device by the virtual source is dissipated in the input and output terminations, and on internal losses of the circuit.

There is quite a simple way of predicting the needed number of iterations implemented in *Amigo* option (see [Section E 2.1.2.1](#)). It is based on the observation that since the FDTD method simulates natural processes in the device, the necessary number of iterations (and the computing time) directly

depends on the loaded Q-factor of the analysed structures. High-Q resonant structures need much longer computing time. On the other hand the Q-factor directly influences the frequency resolution needed to reproduce the characteristic of interest with sufficient accuracy. Thus declaration of the frequency resolution is sufficient to calculate the needed number of iterations. The user is typically well aware of the frequency resolution he requires to make the simulation of the frequency-domain characteristics informative and thus application of this criterion is quite straightforward. A concern may arise that there are structures of a very low-Q but very large size and thus the long simulation time is needed to simulate displacement of the virtual pulse through the structure. However the size of the structure also directly influences the variations of the frequency-domain characteristics and the user can usually easily predict the frequency resolution needed to take into account the "size factor". To make such a prediction easier *Amigo* informs the user how many times the pulse would travel across the structure with the assumed number of iterations. Resuming: we recommend the application of *Amigo* as a basic way of predicting the needed number of iterations, but we also encourage the users to experiment with other criteria listed below.

There are two general criteria for terminating the simulation during its interactive operation:

- During the simulation we can watch the field distribution to find out when the field amplitudes are reduced to negligible values.
- We can also watch the S-parameter curves to find out when their shape stabilises.

Two other criteria are available in *QW-Simulator* for two important categories of problems:

- S-parameter analysis of shielded lossless multiports. *QW-Simulator* integrates power entering and leaving through all the ports. The user can watch the power balance curve, and stop the simulation when power balance becomes equal to unity in the considered frequency band.
- Analysis of radiation patterns and return loss of lossless antennas. *QW-Simulator* calculates radiation efficiency as a ratio of power radiated by the antenna to power delivered to the antenna by the source. The user can stop the simulation when efficiency approaches 100% at the frequencies of interest.

Note that after introducing losses, the simulation time for a particular structure can be reduced. The user sets the number of iterations by application of *Amigo* in the *QW-Editor*, or in the so-called *tasker files* (see [Chapter E 4](#) and [S 4](#)). It is understood that an inexperienced user will start from the *Amigo* option. When the simulation is suspended at the number of iterations calculated by *Amigo* he will resume the simulation to see if the obtained characteristics really converged adequately to the final solution. This will help in setting in the next runs the frequency resolution giving the best results in reasonable computing time. In most cases an experienced user will be able to set easily a correct number of the frequency resolution in *Amigo* or directly the needed number of iterations based on previous experience with the particular types of circuits.

To terminate the subject of needed number of iterations let us note that in the case of structures of very high-Q an optional signal postprocessing module *QProny* can be used to decrease the needed number of iterations.

E 2. Graphical Editor

E 2.1. Main menu and submenus of QW-Editor

After entering into the *QW-Editor* we can see the menu together with submenus. This menu contains six groups of commands.

E 2.1.1. File group of commands

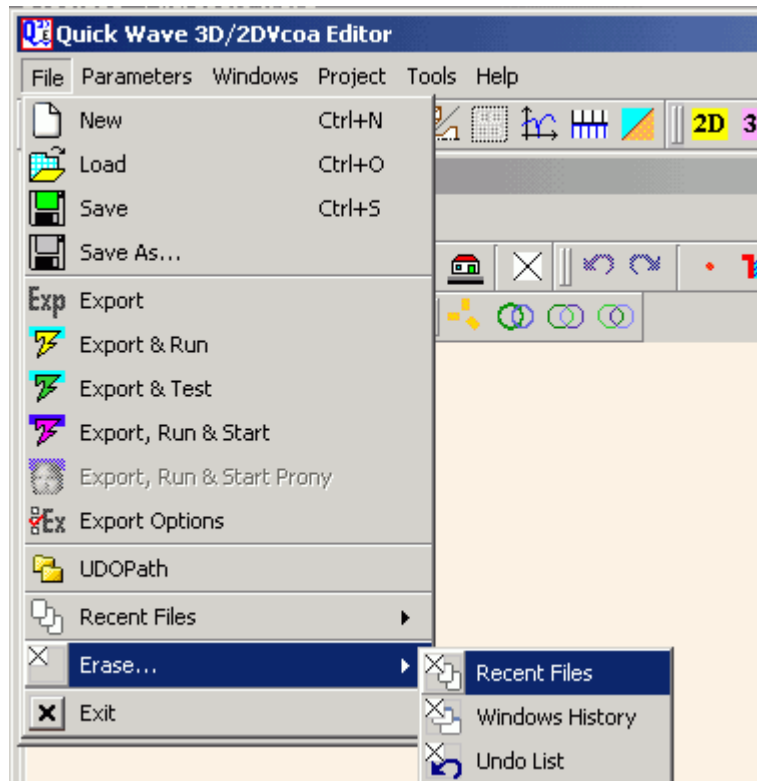


Fig.E 2.1.1-1 File group of commands of QW-Editor menu.

File group contains typical operations (reading, writing) on files describing the structures to be analysed (*projects*). The *projects* are saved on a disk under the name chosen by the user with the extension **.pro*. Beside these commands the group contains also three *Export* commands. *Export* means creating on a disk a set of files describing the *project*, which are needed by the *QW-Simulator*. *Export&Run* command not only performs the export but also opens the *QW-Simulator* as a Windows application and loads *project_name.ta3* and other exported files into it. The simulation can then be started by simply clicking over the green light in the *QW-Simulator*. *Export&Test* performs the same actions as *Export&Run* but loads *project_name.pa3* only and opens *View-TestMesh*. However since other exported files are not loaded this command cannot be followed by starting the simulation. *Export,Run&Start* command performs all actions of *Export&Run*, and then automatically starts the simulation. *Export,Run&Start Prony* command is available if *Prony method* is activated in the *Processing/Postprocessing* dialogue. It performs the same actions as *Export,Run&Start* but simulation is started on file *project_name_prony.ta3*. The user can influence the *Export* process, and thus the subsequent process of circuit analysis, through the *Export Options* command (see [Fig.E 2.1.1-2](#)). In particular:



Fig.E 2.1.1-2 *Export option* dialogue window.

- o *Suppress singularity corrections* means that field singularity corrections will not be applied at metal edges, corners and wires;
- o *Suppress density/SAR* means that density and SAR matrices will not be allocated in *QW-Simulator*. It is a fast way of reducing memory requirements in preliminary simulations of the problems prepared for further thermal simulations;
- o *Allow BHM* is an option available to *QW-3D* users who have additionally acquired *QW-BHM* modules; it instructs *QW-Simulator* to reserve more memory needed for *QW-BHM* operation;
- o *Allow heat flow* is conditional upon *Allow BHM*. It means that heat transfer analysis will be performed at each *BHM* step, if a proper *Heat Flow* module is set in *Set Tools* of *QW-Simulator*, and if all thermal media parameters are defined in media **.pmo* files;
- o *Allow rotation* is conditional upon *Allow BHM*. It means that load will be rotated at each *BHM* step, with rotation parameters set via *Details...* button. It instructs the *QW-Editor* to prepare all files that are necessary to perform simulation of heating of a rotating load; the parameters of the rotation can be set in a dialogue window displayed on screen after *Details* button is pressed.

UDO Path opens *Set UDO Path* dialogue. It allows setting the directories where *QW-Editor* will be looking for **.udo* files, if they are not found at the originally indicated location.

Recent files shows a list of files that have recently been loaded.

Erase provides commands for clearing the history of *Recent Files*, *Windows History* and *Undo List*.

Exit ends the *QW-Editor* session and saves the program environment. On the next entry to the *QW-Editor* we will automatically get the display of the last *project* with the same windows settings as before the *Exit* command.

E 2.1.2. *Parameters group of commands*

Parameters allows setting basic information about the *project* and the desired way of its analysis:

- o *Amigo* opens *Amigo* dialogue with the basic choice of FDTD *Mesh control* in a *Manual* way or by a fully automatic *AMIGO* system. Parameters for the *AMIGO* system are set in the same *Amigo* dialogue. *AMIGO* is a new feature of version 6.0, described in [Section E 2.1.2.1](#).
- o *Circuit type* allows setting dimensionality and basic media environment of the *project*. It is also accessible through the *Setup-Circuit type* path, and it is described in [Section E 2.2.1](#).
- o *I/O Ports* allows configuring ports of the structure; see [Section E 2.5.1](#).

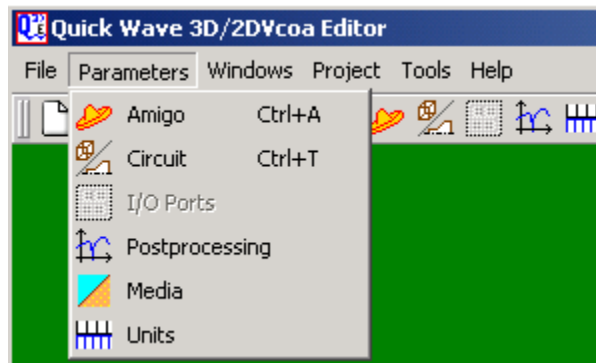


Fig.E 2.1.2-1 Parameters group of commands of QW-Editor menu.

- o *Postprocessing* allows requesting output data (S-parameters, radiation pattern etc.) and modifying frequency range of operation; see [Section E 2.5.2](#),
- o *Media* allows defining and modifying *project* materials; see [Section E 2.5.3](#),
- o *Units* allows selecting units for drawing the structure; see [Section E 2.5.4](#).

E 2.1.2.1. AMIGO

Automatic Meshing Intelligent Generation Option (AMIGO) is a feature introduced in version 6.0. It serves two purposes. Firstly, it optimises the meshing so as to provide requested wavelength resolution in all media while avoiding small cells. Secondly, it allows fast setting of frequency ranges for all ports as well as S-parameter and FD-Probing postprocessings. Additionally, it shows useful information about: details of structure definition that cannot be modelled within requested mesh constraints, time step forced by the current mesh, expected duration of the analysis, and allows setting automatic stop criteria.

We hope that *AMIGO* will be enjoyed and much used by both experienced and novice users. That is why we are placing its basic description at the beginning of the *QW-Editor* Reference Guide. Most of the examples of application of *QW-3D* had been prepared before *AMIGO* was developed. That is why the *User Guide* Sections: [UG 2.2](#) to [UG 2.15](#) or [Sections V2D 2.1-2.3](#) contain examples defined and meshed without *AMIGO*. [Section UG 2.16](#) or [V2D 2.4](#) describes an example with *AMIGO* meshing. We suggest the following learning path of *AMIGO* applications:

- read [Section UG 2.16](#) or [V2D 2.4](#) with possible cross checking of the information provided there with information provided in this Section (E 2.1.2.1).
- try to apply *AMIGO* in other standard examples described in Sections: [UG 2.2](#) to [UG 2.15](#) or [V2D 2.1-2.3](#).

When the geometry of the structure to be analysed is introduced into *QW-Editor* press  (*Parameters-Amigo*) to obtain the dialogue as shown in [Fig.E 2.1.2.1-1](#).

AMIGO dialogue contains four groups of commands

- *Circuit* group of commands is a copy of chosen commands accessible also by the path *Setup-Circuit type* described in [Section E 2.2.1](#). Thus it will not be described here.
- *Frequency* group allows setting the frequency band of interest. The limits of frequency band f_1 and f_2 are to be defined in the upper line of that group. If the checkbox of the line below is checked, these frequency bounds f_1 , f_2 will be copied to the *Parameters I/O Ports* dialogue as the frequencies f_1 and f_2 of the *Excitation waveform*, *pulse of spectrum* $f_1 < f < f_2$. They will be also

copied to the dialogue *Parameters–Postprocessing* setting the bounds of the frequency range for all active *S-parameter* and *FD-Probing* postprocessings. The frequency resolution df will be used to find the minimum number of FDTD iterations needed to obtain well converged results (see *FDTD iterations* group of commands for more explanations). The value of $df/5$ will be used as a frequency step for all active postprocessings.

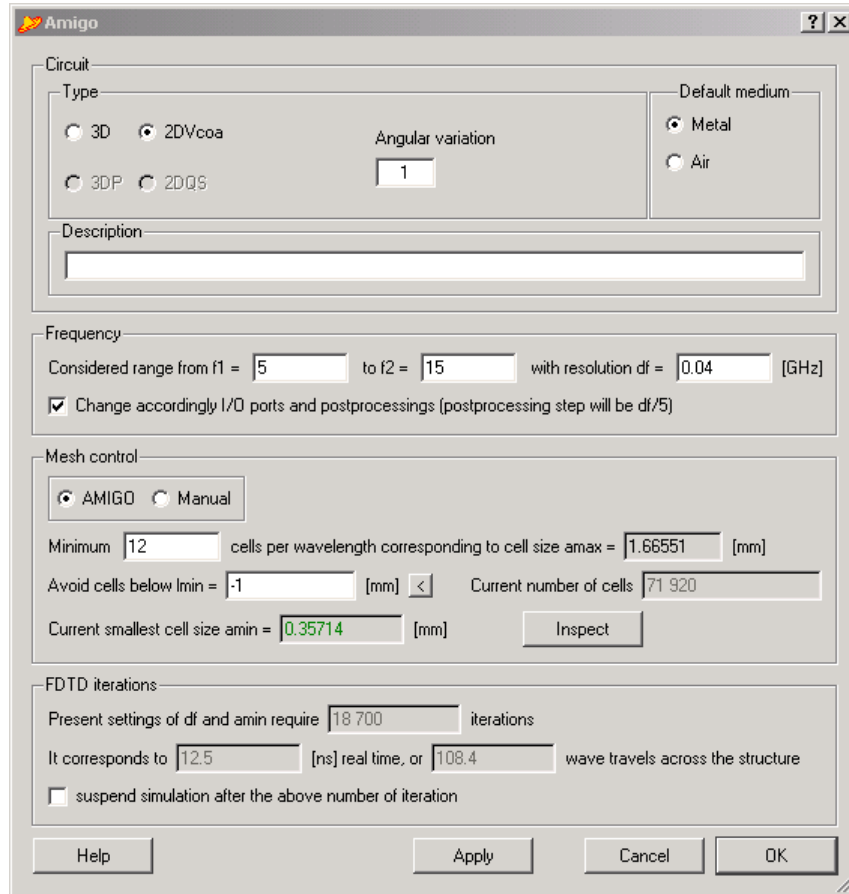


Fig.E 2.1.2.1-1 AMIGO dialogue window.

- *Mesh Control* group allows to switch *on/of* automatic meshing performed by *AMIGO*. It contains two boxes with basic parameters to be set by the user:
 - The first of them is used to set the minimum number of cells per wavelength. *AMIGO* will take this number as a guideline to prepare the meshing in such a way that in all materials used in the *project* the wavelength-to-cell size ratio does not drop below this value. In typical applications this number should be set to 10-15 but some precise calculations may require refined meshing with 20 cells per wavelength or more.
 - The second number $lmin$ in the box: *avoid cells below* gives a suggestion to *AMIGO* to limit the automatic mesh refinements in such a way that too small cells are not generated. Very small cells may have a strong effect on computing time. Not only the total number of cells increases but also the FDTD step in time is automatically adjusted. The box with “<” sign allows to set a relatively reasonable default value when the user is not sure what initial value should be introduced. In particular, it can be used when the software has no hints about $lmin$ and displays value -1 in red. Attention: setting $lmin$ higher than $amax$ should be considered as inconsistent and may produce somewhat confused reaction of *AMIGO*.

The software also displays information about three other parameters, which depend on the above user’s choices. They are: the cell size (in air) corresponding to the chosen number of cells per

wavelength at the highest frequency in the chosen range (f_2), the current number of FDTD cells in the *project* and the smallest cell size in the current FDTD meshing. In general, the smallest cell size should not be smaller than half of the cell calculated for the frequency f_2 in the medium of the shortest wavelength, and not smaller than half of l_{min} set by the user. If it obeys above conditions, it is displayed in green. Otherwise it is displayed in red suggesting that some details of the geometry or *mesh-snapping planes* explicitly enforced by the user make the cell-size smaller. The user is encouraged to press the button *Inspect* to investigate the cause of generating such a small cell. The obtained window shows the mesh-snapping planes generated by the software. They are classified into four groups of priorities (*Master*, *Hard*, *Soft* and *Weak*). The meaning of those priorities is described at the end of this Section.

Other information provided in the window is assumed to be self-explaining.

- *FDTD iterations* is a group providing information about the expected duration of the simulation. In particular it shows:
 - The number of FDTD iterations needed to obtain the assumed frequency resolution df . From the Fourier transform properties, resolution df requires simulation time $T=1/(2 df)$. On the other hand, $T=N*dt$ where N is the number of FDTD iterations and dt is the time step of the FDTD leap frog algorithm. The software automatically chooses dt so that it is proportional to the smallest cell size (a_{min}) to obey the Courant stability criterion. Thus both a_{min} and df influence the required number of iterations.
 - Requested physical time of FDTD simulations $T[ns]$ (see the previous paragraph). This parameter may be informative especially in problems considering transients of pulses.
 - The value of T expressed as the number of wave travels across the structure (assuming that it is filled with air). This value may be informative in the case of non-resonant structures of relatively big dimensions when we should set the simulation time long enough to allow the reflected wave to bounce back a few times, producing well converged simulation results.

Moreover in this group we have one checkbox. It enforces suspension of the simulation process after N iterations (or in other words after time T).

In general, *AMIGO* tries to optimise the mesh taking into account the requested number of cells per wavelength in each medium as well as the geometrical properties of the structure. Although *QW-3D* uses a conformal FDTD method, the accuracy of simulation may be enhanced when the geometrical details are well synchronized with the actual FDTD mesh. This can be obtained by introduction of *mesh-snapping planes*. In general those planes may be of type:

- *E* imposing that the mesh-snapping plane is aligned with the FDTD cell edge where the tangential *E*-field is calculated,
- *M* imposing that the mesh-snapping plane is aligned with the centre of FDTD cell where tangential *H*-field is calculated,
- *N* (neutral) imposing that a mesh snapping plane of either *E* or *M* type is generated in that place.

In most applications, the *E*-type of mesh-snapping planes are preferable because they impose the best conditions for accurate analysis of the most critical parts – edges of metal elements in the structure. Thus in general *AMIGO* tries to generate the mesh-snapping planes of *E*-type along edges of geometrical elements and passing through their vertices. However, in many practical *projects* such a process (if unrestrained) would result in generation of some very small FDTD cells. This would cause a radical drop in the smallest cell size a_{min} and as a consequence in the basic time step dt . This in turn would cause a drastic increase of computing time.

To avoid such an effect, *AMIGO* sets priorities to the mesh-snapping planes and, in case of conflicts, eliminates those less important in order to try to keep the cell size above declared value of *lmin*. There are four classes of priorities of mesh snapping planes.

- *weak* – generated by *AMIGO* and eliminated in the first place when the condition for *lmin* cannot be met.
- *soft* – generated by *AMIGO* and eliminated in the second place when the conditions for *lmin* cannot be met.
- *hard* – generated by *AMIGO* upon the user's declaration of a particular element as *hard* or of *hard edges*. They are not automatically eliminated by *AMIGO* when the condition for *lmin* cannot be met. Instead, they can be reviewed by the user using *Inspect* command in the *Mesh Control* group of *AMIGO* dialogue and possibly deleted upon the user's specific request. *Hard* planes are mostly used for metal edges (like for example edges of a strip of a microstrip line). At metal edges, the fields are highly singular. When a metal edge is aligned with limits of FDTD cells *QW-3D* automatically introduces special mathematical models of field singularities to enhance the modelling accuracy. Because of fast changing fields near such edges it is also advisable to refine the FDTD cells near the edge. That is why the *hard mesh snapping planes* generated by *AMIGO* are accompanied by *weak planes* separated from the *hard planes* by the distance *lmin*.
- *master* – are those *mesh-snapping planes* explicitly declared by the user outside *AMIGO* system. They can also be viewed using *Inspect* command in the *Mesh Control* group of *AMIGO* dialogue.

From the *AMIGO* viewpoint, all the geometrical *elements* of the considered structure can be classified into four groups:

- *disabled* – no action by *AMIGO*,
- *soft* (default) – *AMIGO* generates *soft planes* along edges of these *elements* and at their limits, as well as *weak* planes passing through their vertices,
- *hard edges* – *AMIGO* generates *hard planes* along edges, soft planes along the limits and *weak* planes passing through the vertices,
- *hard* – *AMIGO* generates *hard planes* along edges and *hard planes* passing through the vertices.

The properties of each *element* can be changed manually in the *Edit-Element* dialogue.

E 2.1.3. Windows group of commands

Windows allows controlling the widnows displayed:

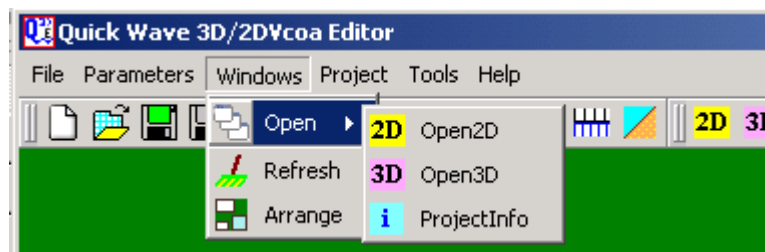


Fig.E 2.1.3-1 *Windows* group of commands of *QW-Editor* menu.

- o *Open* allows opening a new window with three options:
 - ♦ *Open 2D* opens a *2D window* describing a chosen section of the 3D structure,

- ♦ *Open 3D* opens a *3D window* showing the 3D structure in OpenGL,
- ♦ *Project Info* opens a window with basic information about the *project status* (its *name*, *description*, *mesh*, *objects* and a list of *paths* that is helpful in software maintenance and user support). It also contains warning messages issued by *Export* procedures or by *Parser* (interpreting *UDO* commands).

Attention:

- More elaborate description of operations on these windows is presented in the following [Sections E 2.2](#) and [E 2.3](#).
- The user can open as many windows as may be useful for visualisation of the circuit.
- The structure can be defined / modified only in a *2D window* located in the XY plane. This is why such a window is called *2D QW Edit*. All other windows (3D or 2D in other planes than XY) are for visualisation only, and they are called *QW View*. Any changes of dimensions or shape of the designed structure introduced in one of the *QW Edit windows* will automatically appear in all the other displayed windows.
- *Refresh* command shows in all *windows* the most recent state of the *project*, including changes of *mesh* or *local coordinate system*.
- *Arrange* command can be especially useful when many windows are open, and some may have been erroneously pushed out of the screen. Fuzzy appearance of the *project* may result from its moving between two computers working with different resolution. *Arrange* arranges the windows on the screen according to the following rules:
 - ♦ one *2D window* (if open) is moved to upper left corner,
 - ♦ another *2D window* (if open) is placed in lower left corner,
 - ♦ one *3D window* wireframe view (if open) is moved to upper right corner,
 - ♦ one *3D window* solid view (if open) is moved to lower right corner,
 - ♦ windows too big to fit on the screen are reduced in size,
 - ♦ windows pushed out of the screen are pulled back.

E 2.1.4. Project group of commands

Project group contains two commands needed for operation with *biphased UDO* objects:

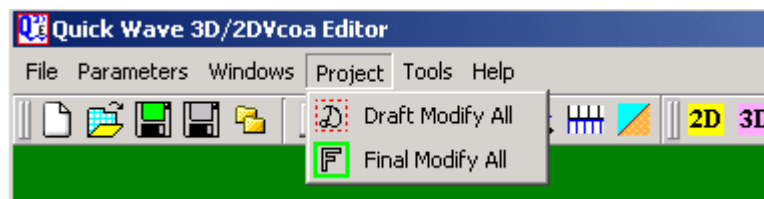


Fig.E 2.1.4-1 *Project* group of commands of *QW-Editor* menu.

- *Draft Modify All* redraws all *UDOs* in *draft* phase,
- *Final Modify All* redraws all *UDOs* in *final* phase.

E 2.1.5. Tools group of commands

Tools group provides access to external applications or format converters. They are:

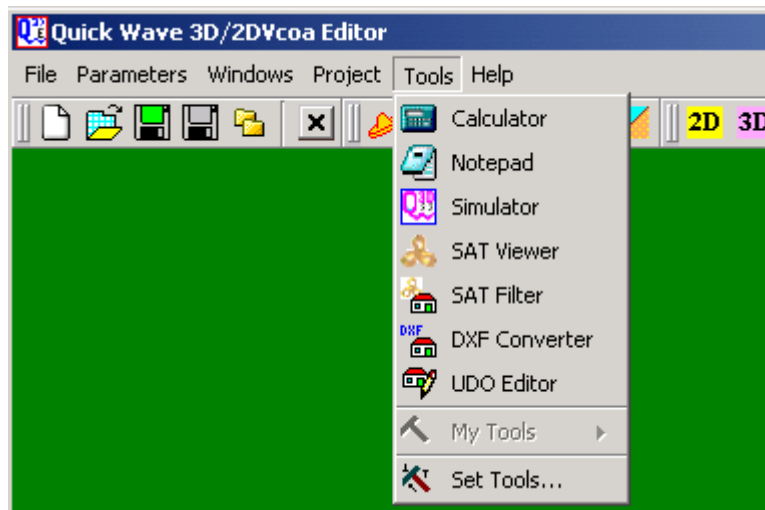
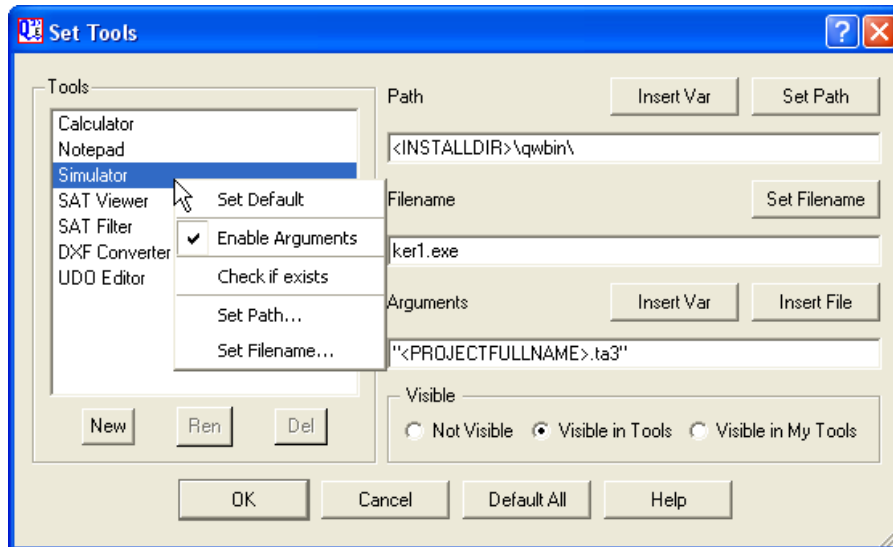


Fig.E 2.1.5-1 *Tools* group of commands of *QW-Editor* menu.

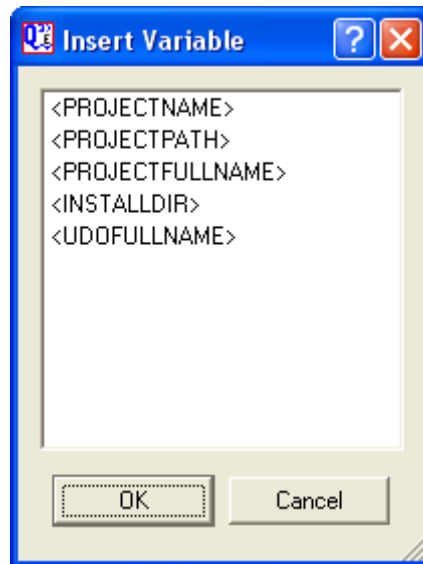
- o *Calculator* – which is supposed to be a standard calculator available in Windows system.
- o *Notepad* – a text editor, Notepad supposed to be the default one.
- o *Simulator* – the *QW-Simulator*; when invoked from here will be started without any reference to the *QW-Editor*. In particular the *Export* operation on the current *project* will not be performed.
- o *SAT Viewer* – This commands causes that the shape of the 3D structure is considered in *QW-Editor* is exported in ACIS SAT format and the free HOOPS 3D Viewer is called to visualise it.
- o *SAT Filter* – This external optional application is used to convert files available in ACIS SAT format into *UDO* format which in turn is read by *QW-Editor* into the current project. Note that the *.sat file must be prepared with a view to its application in 3D EM modelling. It must be internally consistent and must provide a 3D volumetric descripton. In particular, “shell type” geometries are illegal since they cannot be automatically converted into volumes – even if they appear correct in SAT Viewer and analogous viewing tools. ACIS SAT Filter is a joint product of QWED and Vector Fields Ltd.
- o *DXF Converter* – This internal *QW-Editor* option converts chosen files available in DXF format into *UDO* format, which in turn is read by *QW-Editor* into the current *project*.
- o *UDO Editor* – a convenient text editor for *UDO* files.
- o *My Tools* – a group of tools assigned to this group by *Visible in My Tools* option of the *Set Tools* dialogue.
- o *Set Tools* – opens a dialogue for setting new Tools and modifying the existing ones. *Filename* of the executable assigned to each tool is provided, as well as default *argument* where appropriate. *QW-Editor* stores this information in registers and always first looks into registers when trying to pick up a tool. If there is no relevant information in the registers, *QW-Editor* further looks for *zednqw.ini* file from *envir* directory (see [Chapters E 4](#) for more detailed discussion).

Fig.E 2.1.5-2 *Set Tools* dialogue.

Set Tools dialogue allows to the user set the path, filename and argument for the programs or back to default settings via *Set Default* or *Default All* button. There are seven predefined tools:

- o *Calculator* – the default path and filename is *calc.exe* - a standard calculator available in Windows system. There are no default arguments.
- o *Notepad* – the default path and filename is *notepad.exe*- a standard text editor available in Windows system. There default argument is: *<PROJECTFULLNAME>.txt*.
- o *Simulator* – the default path and filename is *<INSTALLDIR>\qwbin\ker1.exe* and the default argument is *"<PROJECTFULLNAME>.ta3"*.
- o *SAT Viewer* – the default path and filename is *C:\Program Files\HOOPS 3D Part Viewer v13.00 for ACIS\bin\nt_i386\acishoops3dpartviewer.exe* and the default argument is *"<PROJECTFULLNAME>.sat"*. The *SAT Viewer* installation is available on the *QuickWave* CD.
- o *SAT Filter* – the default path and filename is *<INSTALLDIR>\bin\ sat2udo.exe* and the arguments are disabled – they are added automatically depending on the options in the *SAT Filter* options dialogue.
- o *DXF Converter* – the default path and filename is *<INSTALLDIR>\qwbin\ dxf2udo.exe* and the arguments are disabled – they are added automatically depending on the options in the *DXF Converter* options dialogue.
- o *UDO Editor* – the default path and filename is *[INTERNAL]* and there are no default arguments.

The user can change path and filename via *Set Path* and *Set Filename* buttons. The user can also insert an internal variable via *Insert Var* button and choose the variable from the *Insert Variable* dialogue.

Fig.E 2.1.5-3 *Insert Variable* dialogue.

The predefined *QW-Editor* variables are:

- o <PROJECTNAME> – returns the filename (without path and extension) of the current project
- o <PROJECTPATH> – returns the path of the current project
- o <PROJECTFULLNAME> – returns the path and filename (without extension) of the current project
- o <INSTALLDIR> – returns the path of the *QuickWave* installation from where the *QW-Editor* was started
- o <UDOFULLNAME> – returns the path and filename (with extension) of the UDO file chosen from the *UDO Library* dialogue ([Section UG 2.8.2](#))

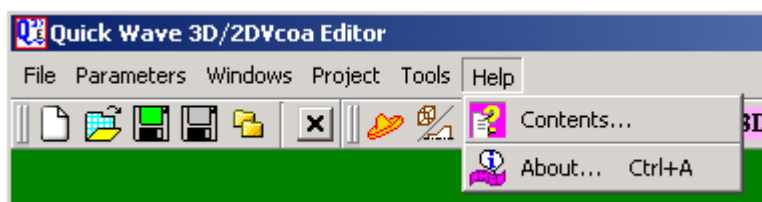
The *Visible* section contains three options:

- o *Not Visible* – the tool will be not visible in the *Tools* menu ([Fig.E 2.1.5-1](#))
- o *Visible in Tools* – the tool will be visible in the main *Tools* menu ([Fig.E 2.1.5-1](#))
- o *Visible in My Tools* – the tool will be visible in the *Tools-MyTools* menu ([Fig.E 2.1.5-1](#))

The user can add, change or delete the tools via *New*, *Ren* and *Del* buttons. Please note that the names of the seven predefined tools can not be changed and this tools can not be deleted.

E 2.1.6. *Help* group of commands

Help group contains two options: *About* shows general information about *QW-Editor*, *Contents* opens the electronic manual prepared in HTML.

Fig.E 2.1.6-1 *Help* group of commands of *QW-Editor* menu.

E 2.1.7. Overview of icons within QW-Editor main window

Below the menu described above we have a toolbar ([Fig.E 2.1.7-1](#)) containing icons, which are shortcuts to the basic commands.



Fig.E 2.1.7-1 The QW-Editor toolbar.

	File-New		Project- Draft Modify All
	File-Load		Project- Final Modify All
	File-Save (Ctrl+S)		File-Export
	File-Save as		File-Export&Run
	File-UDOPath		File-Export&Test
	File-Exit		File-Export,Run&Start
	Parameters-Amigo		File-Export,Run&Start Prony
	Parameters-Circuit type (Ctrl+T)		File-Export Options
	Parameters-I/O Ports		Tools-Calculator
	Parameters-Postprocessing		Tools-Notepad
	Parameters-Units		Tools-Simulator
	Parameters-Media		Tools-SAT Viewer
	Windows-Open-2D		Tools-SAT Filter
	Windows-Open-3D		Tools-DXF Converter
	Windows-Open-ProjectInfo		Tools-UDO Editor
	Windows-Refresh		Help-Contents...
	Windows-Arrange		Help-About...(Ctrl+A)

E 2.1.8. Displays of the shape of QW-3D projects

Example of the main window is presented in [Fig.E 2.1.8-1](#). It presents a coax to waveguide transition considered in ...Wgtocx\wgtocx1.pro example thoroughly described in [Section UG 2.2](#) of QW-3D UG. The project is visualised in four windows. The two on the left are 2D, the two on the right are 3D.

Let us note that in the *QW-Editor* we use in parallel two systems of coordinates. One is the *general 3D* system of the axes X , Y and Z . The other is a *local 2D* system bound to a particular *2D window*. The variables of this system are U and V . The local system can be changed using *Setup-Plane* command. For example, it can be rotated by a particular angle and in such a case the lines drawn parallel to it will be inclined by this angle with respect to the XY axes of the *general 3D* system.

The general coordinate system is marked in the *3D window* by three short thick axes (coloured red, green and blue, *RGB* corresponding to XYZ).

The position of the cursor in each of the *2D windows* expressed in the local coordinates is displayed in the lower part of this window. Here we have the U and V components as well as the increments of these components from a chosen (last click) reference point. These *incremental components* are set to zero after we press the left mouse button. To be able to compare the *local components* with the *general components* it is sufficient to press the space bar and we can see the window with the position of the point in the XYZ coordinate system. Pressing the space bar once again closes the window with XYZ coordinates.

Note that the content and position of the toolbars can be individually set in each window. To show or hide a group of command press the right mouse button to open relevant menu. To change the position click over the grey 10-points field of the particular group of commands and drag it to the desired destination.

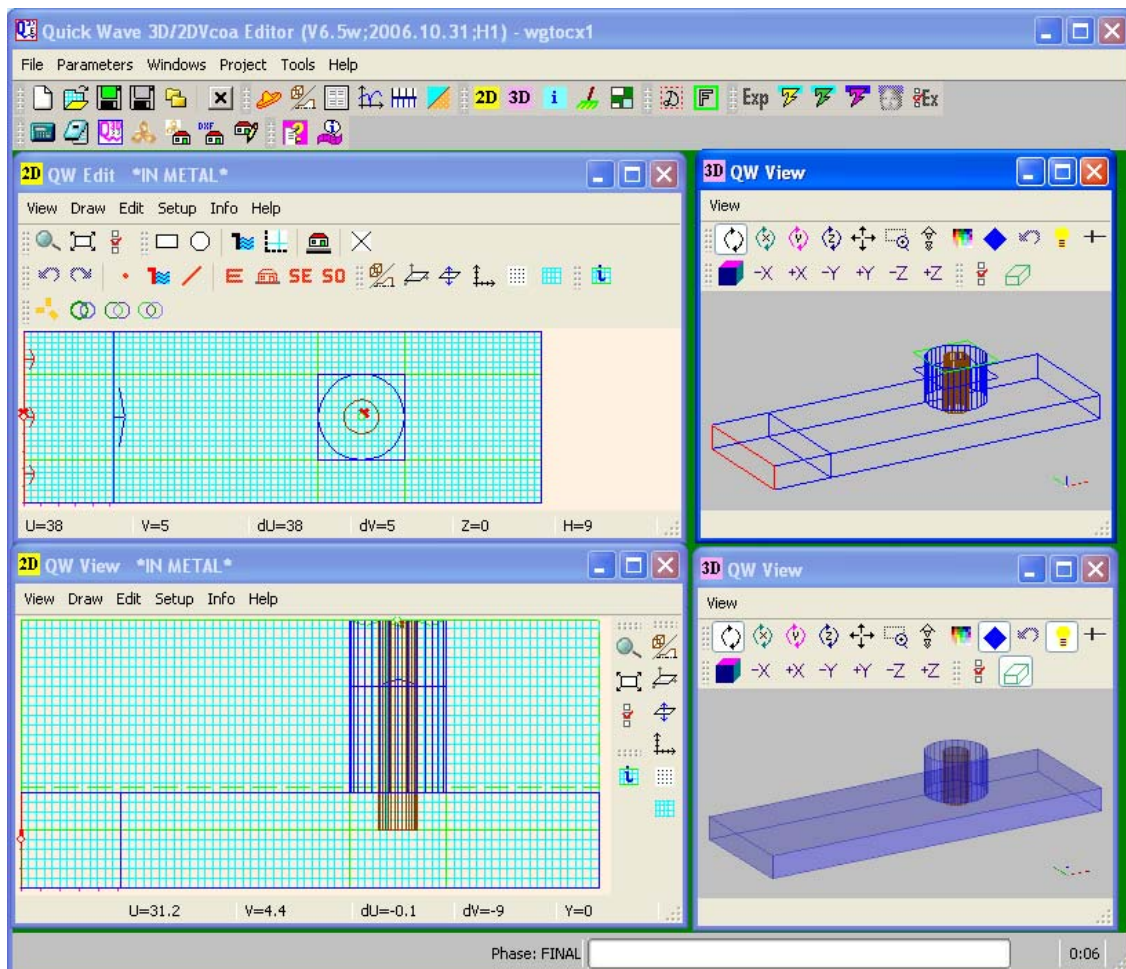


Fig.E 2.1.8-1 Example of the main window of *QW-Editor* with a *QW-3D* project.

E 2.1.9. Displays of the shape of QW-V2D projects

Example of the main window with a QW-V2D standard example `.corhorn\corhorn3.pro` is presented in [Fig.E 2.1.9-1](#). The list of commands and toolbars are basically the same as in QW-3D. Here we will underline the specific features, interpretations and restrictions concerning QW-V2D projects. As it was pointed out in the first chapter of *QW-V2D User Guide* the QW-V2D projects are considered in cylindrical coordinates x, ρ, φ with the assumption that the dependence of fields on the coordinate φ is analytically known. Thus in fact we consider the structure in two dimensions x and ρ . In terms of FDTD meshing we have just one layer of FDTD cells reproducing the shape of the structure in the plane x, ρ (see upper left window in [Fig.E 2.1.9-1](#)). Thus that picture concerns half of the structure long section with the axis of symmetry situated at the bottom of the picture and represented by a thick broken line. The third dimension (with respect to φ) is always equal to 2π and in the 2D windows is symbolically represented by the unitary height of the single layer of FDTD cells. That unitary size does not carry any information but formally can be represented in a 2D window in ρ, φ or x, φ coordinates (see the lower left window of [Fig.E 2.1.9-1](#)). In the 3D windows QW-Editor tries to reproduce the actual 3D shape of the axisymmetrical structure. Although in reality the φ - coordinate always spans over the entire 2π angle (as shown in the upper-right window of [Fig.E 2.1.9-1](#)), it can be visualised also with assumption of the user-defined span of φ (see the lower-right window). Such a visualisation shows better the details of the cross section of the structure.

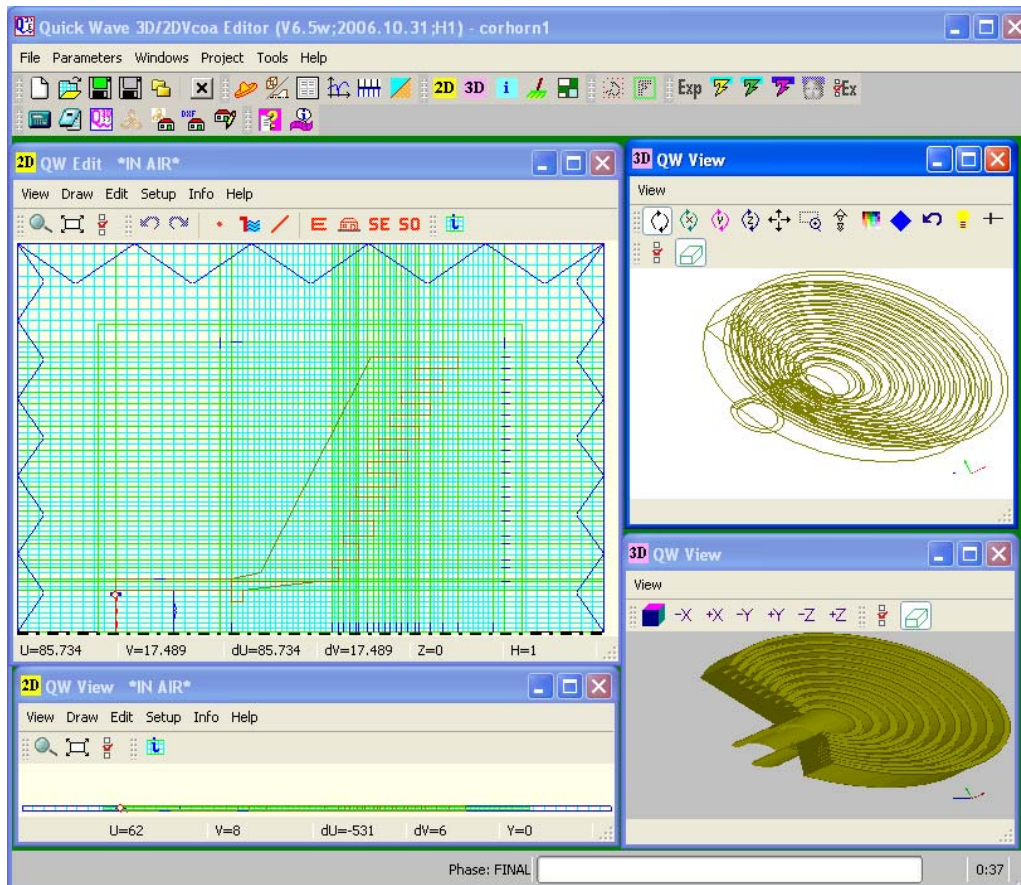


Fig.E 2.1.9-1 Example of the main window of QW-Editor with a QW-V2D project.

Here are some remarks concerning application of QW-Editor in QW-V2D projects:

- basically most features and commands described for QW-3D in x, y, z coordinates are valid here for x, ρ, φ coordinates respectively,

- the single layer of QW-V2D structure must be placed at the level of 0 and have a height 1 (if a structure of different level or height is defined it will be truncated to the levels from 0 to 1),
- in x, ρ plane the structure cannot extend below the axis (ρ cannot be negative).

E 2.2. The 2D windows

Each of the *2D windows* describes a chosen section of the 3D structure. In the case of [Fig.E 2.1.8-1](#) the lower left one describes a section in *XZ* plane (along the waveguide). The upper left window describes a section in the *XY* plane. We will concentrate here on this window because, according to the remarks of [Section UG 2.1](#) the *XY* plane is a natural base for investigating, introducing and correcting the *elements* of which the *project* is composed. It should be noted that when a *2D* window describes the considered structure in *XZ* or *YZ* plane some of the commands of the *Draw* and *Edit* groups are blocked. Thus in practical operation of the software when we have problems with using one of those commands, most probably it is the result of the cursor being focused in a window other than 2D one defined in *XY* plane.

View	Draw	Edit	Setup	Info	Help
Zoom	Undo	Undo	Circuit type	Mesh	
Extents	Redo	Redo	Plane		
Options	Element	Edit Point	Level		
	Combined	Edit Port	Axes		
	Bi-phased Element	Edit Line	Grid/Snap		
	Bi-phased Combined	Edit Element	Coordinates		
	Wire	Edit Object	Mesh		
	IOPorts	Select Element			
	SPlanes/boundaries	Select Object			
	NTF box	Reproduce			
	Plane wave box	Join			
	Get...				
	Object...				
	Clear				

Fig.E 2.2-1 The menu and submenus of a *2D window*.

Let us look at the *2D window* menus shown in [Fig.E 2.2-1](#). There are five groups of commands (apart from help). We will describe them below.

E 2.2.1. The Setup group of commands

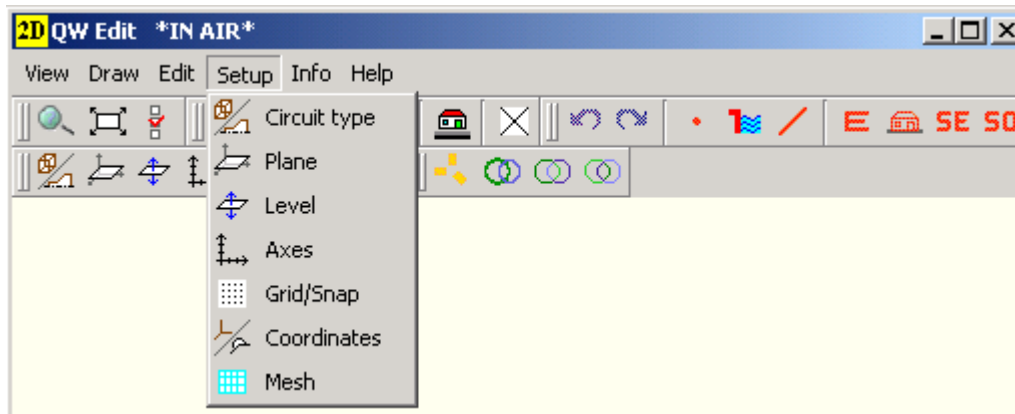


Fig.E 2.2.1-1 Setup group of commands of QW-Edit menu.

The Setup group contains the following commands:

- *Circuit type*. It can also be accessed through the *Parameters-Circuit type* path from the main QW-Editor window. We set here the type of the considered *project*. In a typical application of the package as a three-dimensional tool we put 3D. Then, the *Phase shift per period* box is inactive - it would be needed for 3DP (3D periodic). In the case of 2DVcoa (denoting axisymetrical structures - two dimensional in cylindrical coordinates) we need to define proper angular variation.

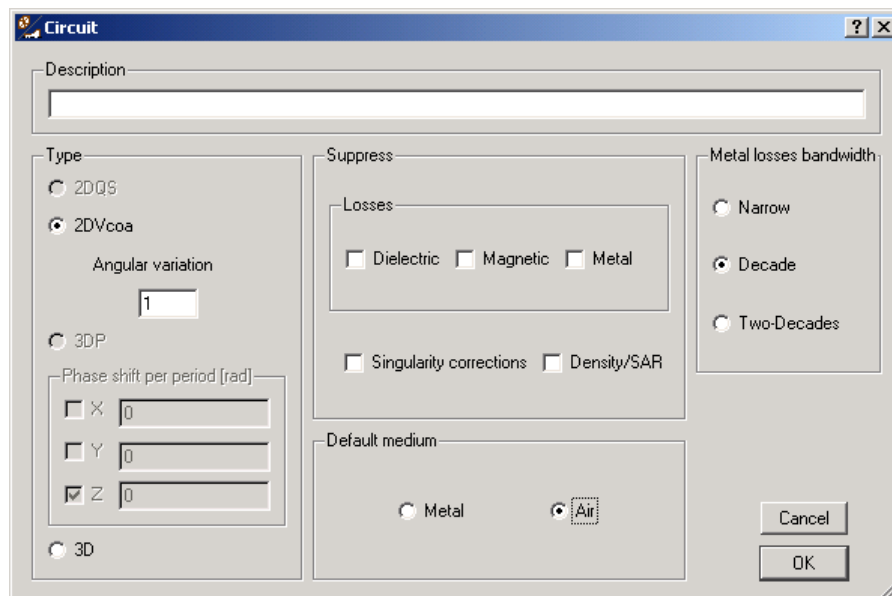


Fig.E 2.2.1-2 Circuit dialogue window.

We further have a choice of the *default medium*. This type of medium will be assigned to any volume in the *project*, which has not been explicitly defined as made of some other medium. We have a choice of two types of *default media*: *Metal* (convenient for shielded waveguide structures) or *air* (which would be convenient for radiation problems). Note the following:

- o The choice of the *default medium* is important and accidental change of it may lead to wrong simulation results. That is why the current *default medium* is constantly signalled in the header of each 2D window where we can see the expression **IN METAL** when the default

medium is *Metal* or **IN AIR** when the default medium is *air*. Moreover, the *background colour* of the window changes (see the *View-Options*). On the entry to a new *project* the *default medium* is *Metal*. Thus if you want to model a free space propagation (like for example in antenna applications) you must change it to *air*. It has been noted that a wrong *default medium* setting is a common user-generated error in *QW-3D* operation. Thus please be careful with it.

- o Regardless of the choice of the default medium the default external boundaries of the FDTD mesh are assumed to be made of *Metal*. To assign to them different properties we must do so explicitly by defining them as *I/O ports* or *Special planes & boundaries*.
- o We also have five *suppress* options, for suppressing *dielectric*, *magnetic* or *metal* losses, *singularity corrections*, and *SAR analysis*. The last two options are also accessible as *Export options* from the main *QW-Editor* window (see [Section E 2.1](#)). Suppressing the losses saves memory and computing time. Thus it is very convenient when we want to obtain faster preliminary simulation results without laborious modification of the entire project.
- o The last group of *circuit type* choices regards the bandwidth for rigorous consideration of frequency-dependent skin effect in lossy metals. It has to be pointed out that although the description of losses in metals by finite conductivity is essentially the same as that of lossy dielectrics, the level of conductivity makes a very significant difference. Due to very high conductivity the wavelength becomes so short that it is practically impossible to mesh the lossy metal volume. Thus instead, in *QW-3D* we virtually attach lossy elements to the magnetic field components tangential to the metal surface. To reproduce the frequency dependence, we attach to each of the tangential magnetic field components an RL ladder composed of a finite number (*K*) of cells. With the number *K* increasing we can model accurately the skin effect in a wider band, but at the price of increasing computer time and memory. That is why we allow the user to make a choice of the model to be used. The three options: *Narrow*, *Decade* and *Two Decades* correspond to the number *K* varying from 2 to 12, with appropriate change in wide band properties of the model. The central frequency for the band is taken as: in the case of sinusoidal excitation - frequency of the source; in the case of pulse excitation - central frequency of the band of postprocessing; in the case of pulse excitation and no postprocessing - 10 GHz.
- *Plane*. Here we define the *local 2D coordinate system*. There are two vectors *U* and *V* to be defined. We use the *general system* (*XYZ*) of coordinates to define the common origin and the end of each of the vectors *U* and *V* describing the local system. There are also shortcut keys to set the vectors *U* and *V* in the basic planes of the *general system*: *XY*, *YZ* and *ZX*.

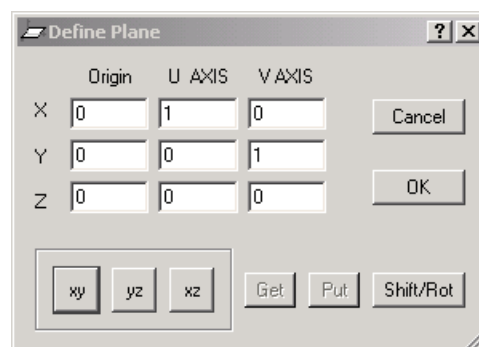


Fig.E 2.2.1-3 *Define Plane* dialogue window.

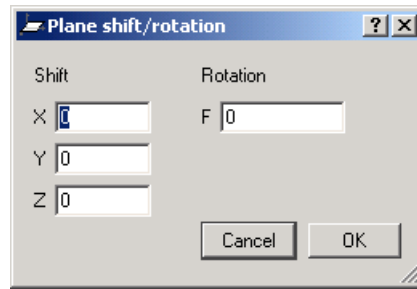


Fig.E 2.2.1-4 Plane shift/rotation dialog box.

- *Level*. Here we can move the *level* of the *local coordinate system* (we could do it also using *Plane* command but in a little more laborious way). Moreover, we can set here the default *height* of the next *element* to be introduced in the *QW-Editor*. Note that the *level* as well as the current default *height* of the *element* to be created are displayed in the lower part of each *2D window* together with the cursor position expressed in the *local coordinates*. Note, that this option is irrelevant for *QW-V2D*, where the default *level* of 0 and *height* of 1 are fixed. Below we present the options available in the *Level* window.

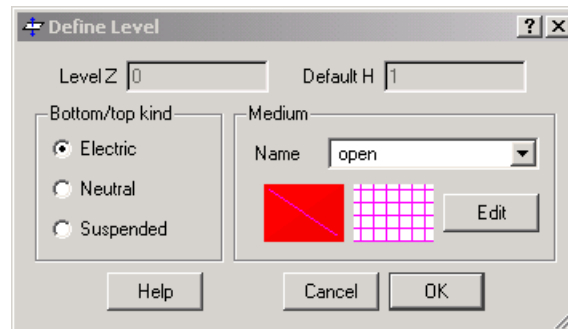


Fig.E 2.2.1-5 Define Level dialog box.

- default medium filling of the currently drawn element,
- bottom/top kind of the currently drawn element. This field indicates if the bottom and top of the element will be:
 - ♦ *Electric* – will enforce the *E-plane* of FD-TD meshing as discussed in [Section UG 2.1](#),
 - ♦ *Magnetic* – will enforce the *H-plane* of FD-TD meshing as discussed in [Section UG 2.1](#),
 - ♦ *Suspended* – will enforce a plane suspended between those considered above.
- *Axes*. Here we decide how the *local coordinate system* should be displayed in the *2D window*. Note that the *2D window* is composed in a way to include the entire *project* and the *axes*. Thus if the axes are too long or incorrectly situated, the *project* may occupy only a small part of the window and be poorly visible.

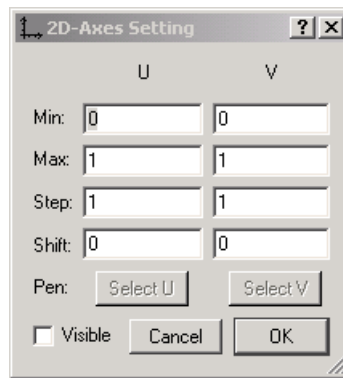


Fig.E 2.2.1-6 2D-Axes Setting dialogue window.

- *Grid/Snap*. By defining the *grid* we set an array of regularly spaced points to be displayed in the *2D window*. They can be helpful in defining the shape of the structure directly on the screen. When the *snap* is *on* any point considered in the *2D window* adopts only coordinates discretized with the *snap* value as the basic increment. It is strongly recommended that the *snap* is always “*on*” and its value is not lower than the accuracy with which the dimensions in the *project* are defined. If the snap is “*off*” two faces of adjacent *elements*, which are seemingly adherent, may in reality form very thin slots, which make the analysis slower and less accurate, if not erroneous. Similar effects may appear for example if we know that the dimensions are defined with 10μ accuracy but use the *snap* of 1μ .

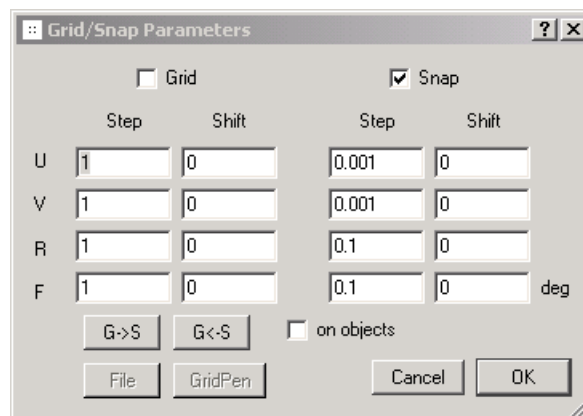
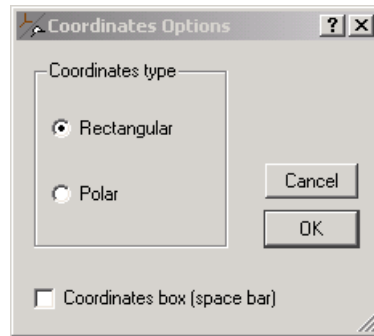
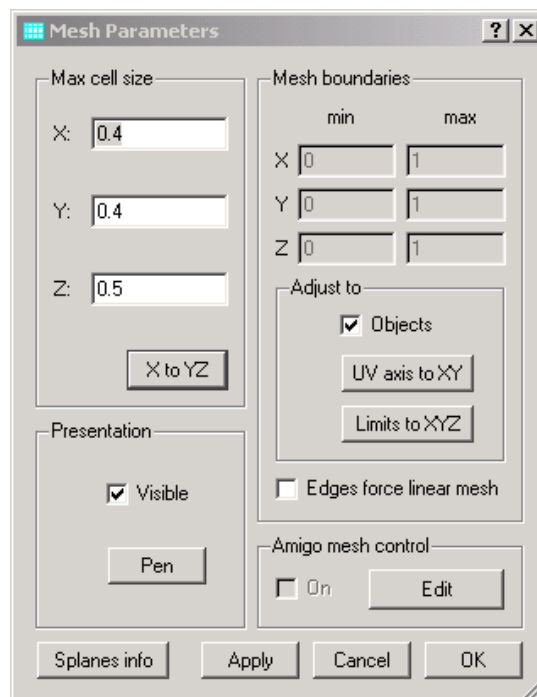


Fig.E 2.2.1-7 Grid/Snap Parameters dialogue window.

- *Coordinates*. This command allows to change the local coordinates from *rectangular* to *polar* ones. Note that *polar coordinates* will only be used for girding / snapping, and displayed as local coordinates (R, F instead of U, V). Drawing and mesh generation are always performed in *rectangular coordinates*.

Fig.E 2.2.1-8 *Coordinates Options* dialogue window.

- *Mesh*. Here we define the parameters of the FDTD mesh (see [Fig.E 2.2.1-9](#))
 - In the right part of the dialogue box we state in which part of the circuit the FDTD mesh is to be created. Typically we want to mesh the entire volume occupied by our structure. Then we shall keep in the section *Adjust to* the default setting: *Objects* checked on. However, sometimes we may wish to analyse only a part of the structure (for example, to speed up the simulation by exploiting spatial symmetry or to analyse only one selected discontinuity with refined meshing). In such a case, we should uncheck the *Objects* box, and set the mesh boundaries explicitly. The *UV axis to XY* box can be used as one shortcut, introducing the *X* and *Y* mesh limits equal to the *2D-Axes Setting* of the active *2D window* (which is itself accessible via *Setup-Axes* command). The other shortcut is *Limits to XYZ*, which introduces the *X*, *Y* and *Z* mesh limits equal to the outermost positions of all the defined *elements*, *objects*, and *ports*. **Please make sure** that all the declared ports are situated inside the meshed part. Otherwise it will be impossible to perform the simulations.

Fig.E 2.2.1-9 *Setup-Mesh* dialogue window.

- In the left part of the dialogue box we can set the maximum FDTD cell sizes along the axes *X*, *Y* and *Z* when we decide not to use *Amigo* option. The *XtoYZ* box copies the *x*-step into the *y*- and *z* - ones. Note that the *QW-Editor* will try to put an integer number of cells between each

pair of the neighbouring *mesh snapping planes*. It will assume as a precondition that any dimension of the cell cannot exceed the maximum size defined here. Thus the maximum cell size should be chosen very carefully. It will play a crucial role in finding a reasonable balance between the conflicting requirements for high accuracy and reasonable computing time, taking also into account the limits of the computer memory.

- o *Presentation visible* permits to display or hide the meshing. *Pen* can be used to modify the *width* and *colour* for mesh lines.
- o *Edges force linear mesh* is an advanced option for gradual change of the meshing. To fully understand how it works, let us summarise some important pieces of information. First of all, *edges* (understood as outermost boundaries of the circuit) are implicit limits of the mesh, and as such they act as the outermost *mesh snapping planes*. However, we cannot edit the *edges* to assign to them any particular *forced cell size* (which we can do for explicitly available *mesh snapping planes*). The *Edges force linear mesh* option alleviates this difficulty. When it is checked on, *edges* actively force the *Max cell size* (defined in the *Mesh Parameters* dialogue, [Fig.E 2.2.1-9](#)) towards the inside of the circuit.

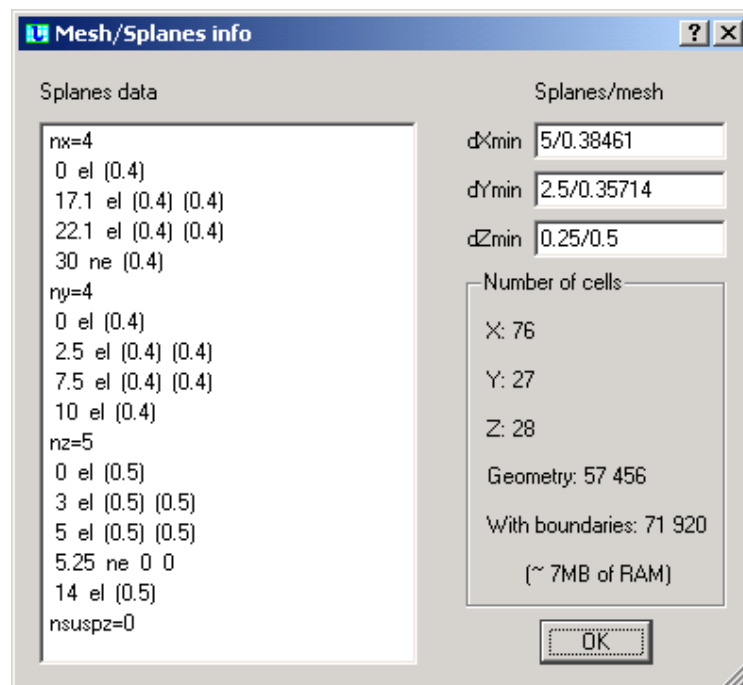


Fig.E 2.2.1-10 Mesh /Splanes info dialogue window.

- o *Splanes info* permits to see the list of all *mesh snapping planes* (also called *Special planes*) created in the *project*. The window of [Fig.E 2.2.1-10](#) appears. The same window can be invoked using *Info-Mesh* command. On the left hand side of this window we can see all the *special planes*, which may have been created in the following ways:
 - ♦ set explicitly by the user (via the *Draw-SPlanes/boundaries* command),
 - ♦ implicitly enforced by *edges* (outermost boundaries of the circuit),
 - ♦ enforced by *ports* (when the options *snap mesh to port plane* or *snap mesh to port circumference* are checked, see [Fig.E 2.4.1-1](#)),
 - ♦ introduced automatically by the software (at the *level* of *bottom* and *cover* of each *element*). Note that only *mesh snapping planes* perpendicular to the Z-axis can be created in this way. If the *default medium* is *Metal*, all these planes will be of *electric*

type. If the *default medium* is *air*, metal *elements* will force *mesh snapping planes* of *electric* type, and non-metal *elements* - of *neutral* type

- ♦ introduced automatically by *Amigo* (if active) in order to optimise meshing.
- The information visible in the *Mesh/Splanes info* dialogue of [Fig.E 2.2.1-10](#) includes:
 - ♦ Location of each mesh snapping planes.
 - ♦ Type: *el* denotes *electric* or *short*, which forces a plane of tangential electric fields (and thus contains the cell sides); *ma* denotes *magnetic* or *open*, which forces a plane of tangential magnetic fields (and thus bisects the cells); *ne* denotes *neutral*, which may coincide with either tangential electric or tangential magnetic fields.
 - ♦ Cell size forced in the negative and positive directions from the *special plane*. If the user has explicitly defined the forced cell size when drawing or editing this particular *special plane* (see [Section E 2.4.2](#) for details), this value will be shown without brackets, and supersede all defaults. However, if the user has not made such an explicit setting, the *forced cell size* will be inherited from the settings of other *special planes*, and it will be shown in brackets. If the *forced cell size* has been explicitly defined for two *special planes* located in the plus- and minus- direction from the considered plane, respectively, the forced cell size for this considered plane will be calculated by linear interpolation. This opens way to the construction of a linearly varying mesh. Remember that the edges of the circuit take part in the algorithm if the *Edges force linear mesh box* is checked.
 - ♦ In the upper right part we get minimum values of the distances between the neighbouring *special planes*, versus minimum cell size in each direction. If the distance between *special planes* is much smaller than the cell size declared in the window of [Fig.E 2.2.1-10](#) (and in particular, if it is equal to the minimum cell size in a particular direction), the software will introduce there a relatively small cell and reduce the time step to assure stability on this small cell. This will significantly slow down the process of the analysis. To warn the user about the existence of such closely situated special planes, they will be marked in *2D windows* by red points at the external contour of the grid.
 - ♦ In the lower right part of the *Info-Mesh* window we can see how many cells have been created in the *QW-Editor* and how many of them will appear in the *QW-Simulator* (including the cells needed to model the boundary effects). This number is crucial for judging if we have sufficient amount of memory to run this example with this meshing, and how long the simulation will take.

Note that the user is strongly recommended to verify the actual settings of the mesh and *special planes* (in the dialogue of [Fig.E 2.2.1-10](#) or in the dialogue of *Amigo*) before running the simulation.

E 2.2.2. The *Edit* group of commands

The *Edit* group has the following commands:

- *Undo* command permits to revoke up to N moves in *QW-Editor*. Here are some remarks regarding the choice of N :
 - N can be set by the user in *zednqw.ini* file (cf. [Section E 4.1](#)).
 - The default value used in the installation of *QW-3D* is $N=10$ (cf. [Section UG 1.4](#) or [V2D 1.4](#)).
 - Note that each remembered move of *QW-Editor* must be stored in the operational memory. Thus do not set high values of N when *QW-Editor* and other applications are executed in parallel, close to the memory limits of your computer.

- o The *undo* record does not register all the actions of *QW-Editor*. Some actions were considered of less importance, and not worth the time and memory expenditures. However, if you have a different opinion, please do not hesitate to contact QWED with comments. The *undo* command is a new feature, and the users' feedback will be valuable for its further enhancement.

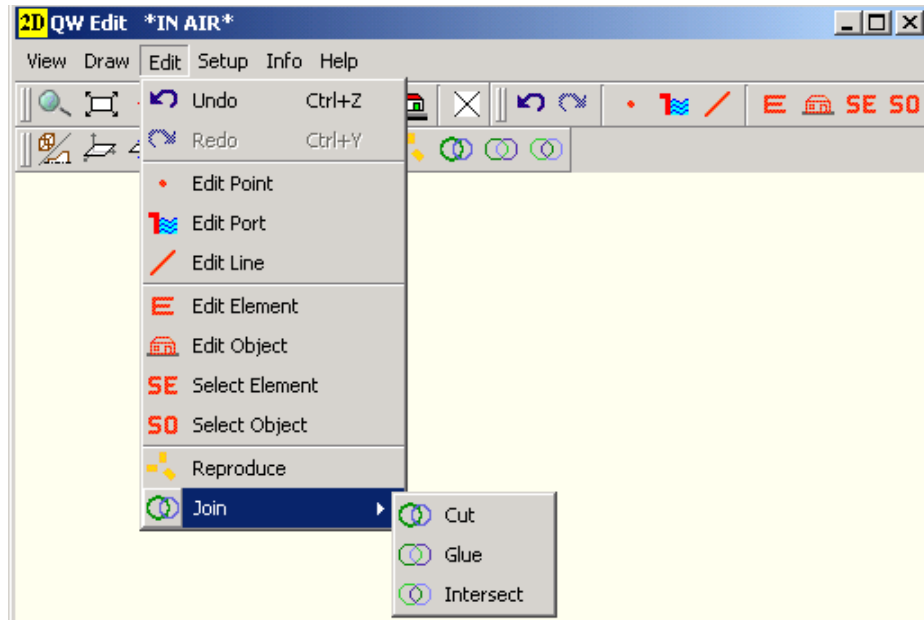


Fig.E 2.2.2-1 Edit group of commands.

- *Redo* revokes up to *N* *undo* commands.
- *Edit Point*. To move a *point* belonging to an *element* we must move the *local coordinate system* to the correct *level*. This can be done by using *Setup-Level* or *Edit-Select Element* commands. After this is done, invoking *Edit-Point* command changes the cursor mode (square cursor) and we are ready to change the position of any *point* belonging to any *element* having the base at this *level*. To do this we approach the cursor to the *point*, click the left button and move the cursor. The *point* is being moved with the cursor. Pressing the left button again will stabilise the new position while pressing the right button will cause its return to the original position. We can also press **K** to be able to introduce the new position of the point using the keyboard. Note that to get out of the *Edit-Point* mode we can press **E** on the keyboard.

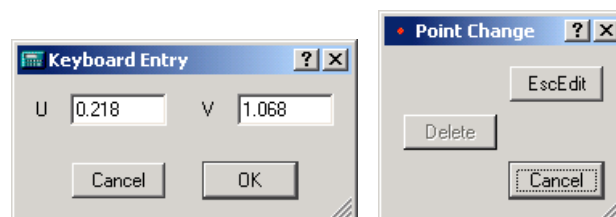


Fig.E 2.2.2-2 Keyboard Entry and Point Change windows.

- *Edit Port*. This command is used to move a *port* or change its parameters. Note that to get out of the *Edit-Port* mode we can press **E** on the keyboard.

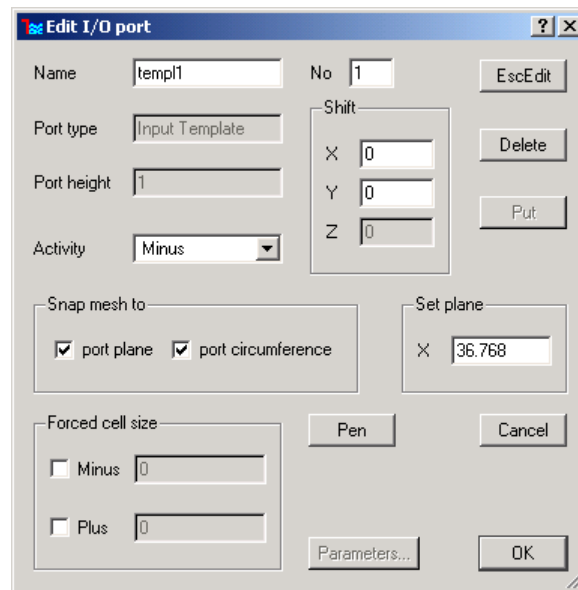


Fig.E 2.2.2-3 Edit I/O port dialogue window.

- *Edit Line*. Invoking this command, and then pressing the left mouse button, selects a line defined at the current *level*, with centre closest to the cursor position. Pressing the right mouse button we see a dialogue window with the choice of several operations:

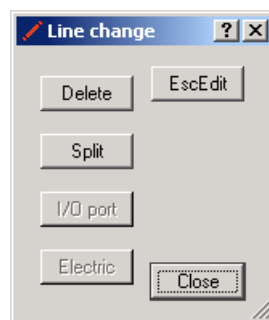


Fig.E 2.2.2-4 Line change window.

- o *Delete* eliminates this line from the contour of the *element*.
- o *Split* produces an extra point in the middle of the line. The position of the new point can be changed using *Edit-Point* command.
- o *Close* abandons the currently selected line (but we remain in the *Edit-Line* mode, with square cursor, and can select another line).
- o *Esc.Edit* abandons the currently selected line and abandons the *Edit-Line* mode (cursor returns to arrow).
- o *Electric* is active only for lines perpendicular to either x- or y-axis. It defines a *mesh snapping plane* (of *Electric* type) along this line.
- o *I/O port* is active only for lines perpendicular to either x- or y-axis. It invokes the same dialogue as the *Draw-I/O Ports* command. However, *Perpendicular to XY plane* is the only available *Orientation* and pressing *Draw* automatically creates the port along the selected line.

- *Edit Element*. This command serves to change the position and parameters of the *elements*. When this command is invoked, we should approach the *element* to be changed with the cursor and press the left mouse button. We can move the *element* and set its new position by pressing the left button once more. We can also press the right button to get into the *Element Change* dialogue (corresponding to this particular *element*) in which its parameters can be changed. The *Element Change* dialogue for a simple element is shown in [Fig.E 2.2.2-5](#); the dialogues for *ports* are described in [Section E 2.4.1](#), and for *wires* and *wire grids* in [Section E 2.2.7](#). Most of the parameters are self explaining. Let us only comment on some aspects of them:

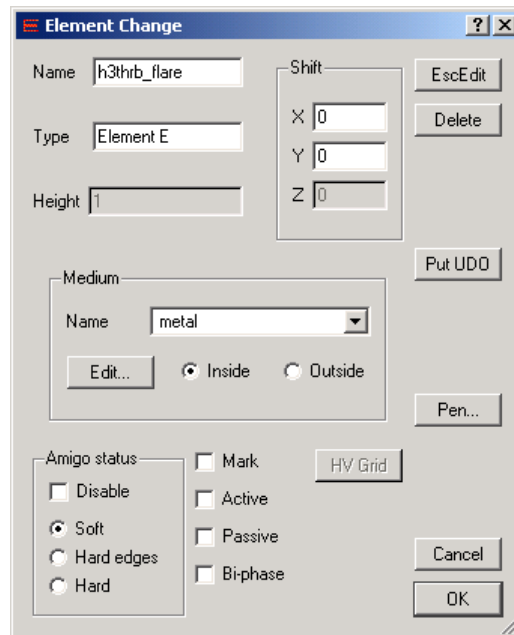


Fig.E 2.2.2-5 *Element Change* dialogue window.

- o Note that the medium can be defined *inside* or *outside* the element. This means that after defining the shape of the *element* the *QW-Editor* acts on the inside (outside) space and fills with the given medium all the available space not occupied by other *elements*. **Note that the recommended way of software operation is to always use the medium inside option!** The *medium outside* option is necessary only in one special case: making holes in metal layers of zero thickness.
- o *Put UDO* command designed to save an *element* on disk as a *UDO object*.
- o The *type of element* is displayed here only as an additional information to the user.
- o Note that when we are in the *Edit-Element* mode the cursor changes to a square one. To return to the *normal mode* (arrow cursor) we must either press **E** on the keyboard or press the right mouse button and then *EscEdit*. It is important for the user to develop a habit of leaving the *edit mode* immediately after the edit operation has been performed. The operation of other commands while in the edit mode can lead to unwanted changes in the *project*.
- o It is possible to *Mark* one or more elements. The *marked elements* form a group, which can be moved or reproduced using the *Edit-Reproduce* command.
- *Edit Object*. This command allows to move the selected *object*, delete it or change its parameters. When this command is invoked the software looks for the *object* closest to the cursor and having the base at the same *level* as the cursor. If we are not sure about the position of the base of the *object* we should use the *Edit-Select Object* command. In the case of *external objects* we can use

the command *Modify* that will open the window with the object's parameterised dimensions. We can change these dimensions and draw the *object* again.

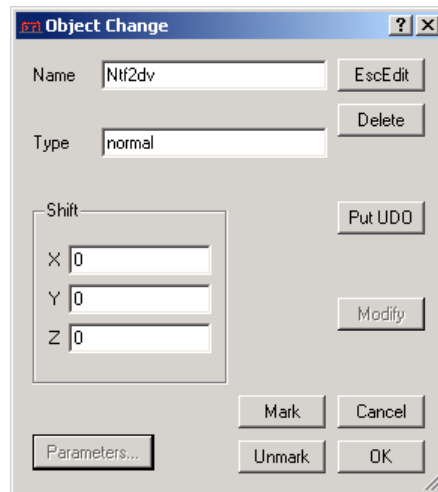


Fig.E 2.2.2-6 *Object Change* dialogue window.

- *Select Element*. This command is used when we want to pick up an *element* on which some operation should be performed. We get a dialogue window in which all the *elements* of our *project* are displayed. For example in the case of *wgto cx1* project (see [Fig.E 2.2.2-7](#)) we have seven *elements*: waveguide, inner and outer conductors of the coax line and four *elements* describing the *I/O ports (templates)* and *reference planes* for S-parameter calculation. Here are the typical operations which can be performed using this dialogue window:

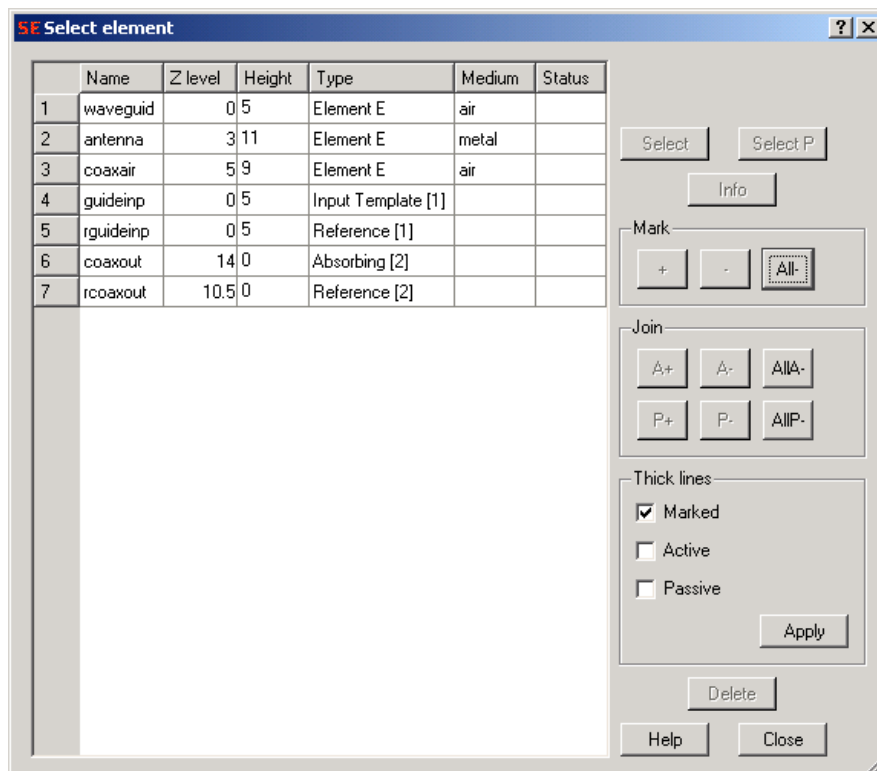


Fig.E 2.2.2-7 Example of *Select element* dialogue window displayed for the case of *wgto cx1* project.

- To move an *element* in space (in *XY* plane) we must move the *local coordinate system* to the plane of the base of this *element*. This can be done by a double click over an *element* on the list or by a single click over the *element* and a click over the *Select* box. The dialogue window disappears and we are back in the *2D window* on a correct *level*. Now it is sufficient to click over the *element* to start moving it to a different place. Sometimes we want to change a position of just one *point* of the *element*. In this case after selecting the *element* we should click over the *Edit-Point* icon in the *2D window* toolbar and choose the point to be moved.
- The operation of selecting an *element* is also convenient when we want to edit it. When the *element* is selected it is sufficient to press the right mouse button to enter the *Element change* dialogue window. [Fig.E 2.2.2-5](#), bottom left, shows this dialogue for a *QW-V2D* antenna *element*. We can change numerically shift the *element's* position (in the case of *QW-3D* the shift would be possible also along *Z*-axis). We can also modify the medium of which it is composed (through *Edit* button) and the colour used for its display. The *HV Grid* button of the *Element change* dialogue becomes active only for metal elements of zero height, and then can be used to convert thin metal layers to horizontal wire grids (see [Section E 2.2.7](#) for more details). Note that invoking *Edit-Element* command through the *Select element* is comfortable especially when we have several *elements* situated in one plane very close to one another (or even interposing) and it is difficult to pick one of them up by mouse operation on the screen.
- It should be noted that each of the *combined elements* (if present) is displayed on the list as a composition of two *elements* of the same name describing the lower and upper plane of the *element*. The types of these *elements* indicated on the list are *bottom* and *cover*, correspondingly. Each of them can be selected separately. However, if you move or modify *bottom*, *cover* will be moved/modified automatically in the same way. On the contrary, *cover* is moved and modified without influencing *bottom*. *Cover* has nominal height equal zero while *bottom* has nominal height equal to the actual height of the *combined element* (and equal to the difference in the *levels* of the *bottom* and the *cover*). However, remember that both these values are purely nominal, and the only way to change the height of the *combined element* is by changing the *level* of its *cover*.
- Each *element* can be *marked* or *unmarked* using the “+”, “-” and “all” boxes. When an *element* is *marked* a letter *M* appears by its name. On the graphical display, *marked elements* will be distinguished from *unmarked* ones by thick line contours. The *elements* are *marked* to point out those, which should be subject to the subsequent *Edit-Reproduce* operations.
- Each *element* can be also defined as *passive* or *active* for *Edit-Join* operations. When this is done a letter *P* or *A* appears by the *element's* name. When performing the *Edit-Join* operations, the *active* *elements* will dominate over the *passive* ones in the conflicting areas.
- Normally the *Select Element* command changes the *local 2D coordinate system* of active *2D window* to the base of the *element* in the *XY* plane. If we wanted to move the excitation point in a port perpendicular to the *XY* plane, we would prefer to have the *local 2D system* situated in the plane of the *port*. This can be achieved by selecting the *port* and then clicking over *Select P(lane)*.
- *Select Object*. This command permits to select one of the *objects* (groups of *elements*) defined in the *project*. A list appears which contains all the drawn *internal* and *external objects*. Each may be followed by status letters: *E*, *O*, *C* denoting that the *object* is: *empty* (contains no *elements* or *objects*), *open* (*elements* or *objects* can still be added), *current* (*elements* or *objects* are currently added to this particular *open object*). After selecting the *object* the program automatically executes the *Edit-Object* command. Note that the *external objects* are parameterised and their dimensions can be modified. The *internal objects* are simply groups of *elements*, which can be moved together.

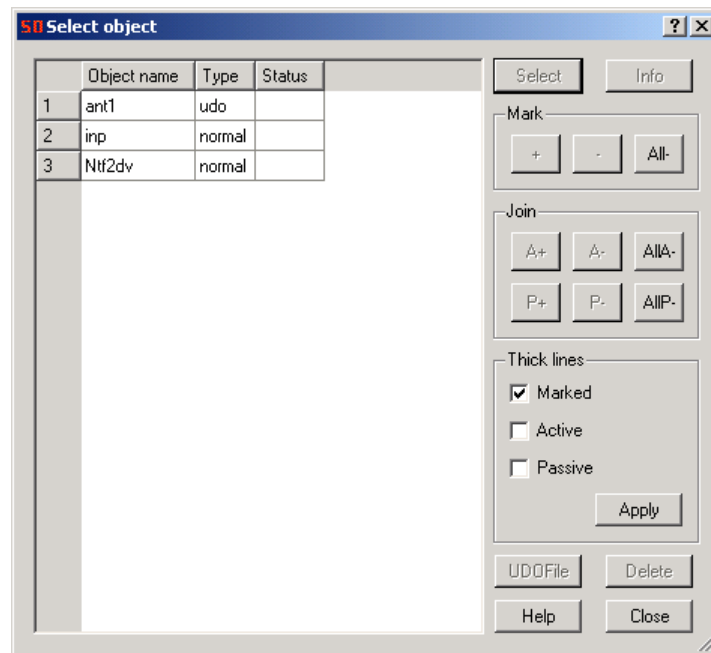
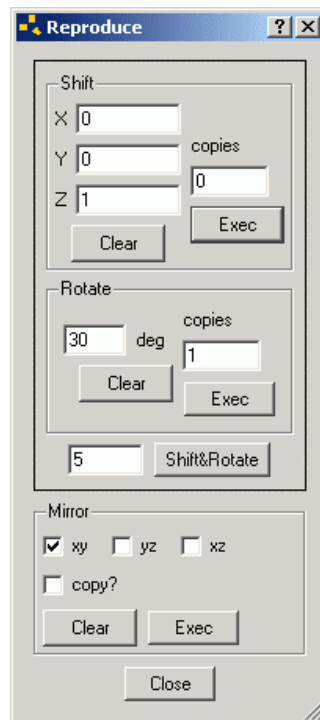


Fig.E 2.2.2-8 Select object dialogue window.

- *Reproduce*. This command operates on the group of *marked elements*. These *elements* are drawn with thick contours. Elements can be *marked* or *unmarked* in the *Select element* dialogue window. *Marked elements* can be shifted, rotated or reflected. The number of copies indicates how many times the operation should be performed. Any number greater than zero leaves also one copy of the *marked element(s)* in the original position. For example, with the settings presented in [Fig.E 2.2.2-9](#) we can produce the following effects:
 - *Exec.* in the *Shift* zone will move the *marked element(s)* by one unit in the *z*-direction.
 - *Exec.* in the *Rotate* zone will make a copy of the *marked element(s)* rotated with respect to original by 30 degrees. Note that the axis of rotation is the point of the last left mouse click over the active 2D window.
 - Clicking over *Shift&Rotate* will produce 5 copies of the *marked element(s)* each one shifted by one unit in *Z* direction and rotated by 30 degrees. Note that one possible application of the *Shift&Rotate* option may be drawing of spiral conductors.
 - *Exec.* in the *Mirror* zone will create a mirror image of the *marked element(s)*, with respect to the *xy*-plane.
 - *Clear* in any zone clears the settings introduced in that zone.

Attention:

It is the user's responsibility to avoid producing intersecting elements during this operation. If such elements are produced, they should be treated using Join option prior to the FDTD simulation. Otherwise they may cause incorrect calculation or even a run time error of the software.

Fig.E 2.2.2-9 Dialogue window for *Reproduce* operation.

- *Join*. This command concerns putting together intersecting *elements*. Prior to its execution we must declare the intersecting *elements* as *active* or *passive* in the *Select element* dialogue window. We have three options of *join* operations seen in the *Join* dialogue window (see [Fig.E 2.2.2-1](#)):
 - *Cut* option we cause the *active element* to make a hole in the *passive* one. *Active element* will not be changed.
 - *Intersect* option is used to produce one *element* of the shape being the common part of the *passive* and *active element* filled with the medium of the *active* one.
 - *Glue* option is used to merge the elements. The resulting *element* will be filled with the medium of the *active* one.

Attention:

When executing the commands of the *Edit* and *Draw* groups we can enter the desired position of the cursor from the keyboard. To do this we should press **K** and use the appearing dialogue box to enter the desired coordinates.

E 2.2.3. The *Draw* group of commands

The *Draw* group contains the following commands:

- *Undo* and *Redo* are equivalent to the *Edit-Undo* and *Edit-Redo* commands (see [Section E 2.2.2](#)).
- *Element*. To draw an *element* we define the shape of its base. The *element* is being introduced on the current *level* and with the current *height*. To change *height/level* settings please use *Setup-Level* command. There are two basic shapes of the element's base:
 - *Rectangle*. To draw a *rectangle* please click the left mouse button two times to mark its corners.

- *Circle*. The circle is approximated by a regular polygon with the number of sides (*sectors*) defined by the user. A high number of sectors gives better accuracy but slows down the QW-Editor operations. The choice of the number of *sectors* gives also a possibility of using the *Circle* command to introduce arbitrary polygons. For example if we want to introduce an irregular octagon we first define a *circle* with 8 *sectors* and then use *Edit-Point* command to move each of the corners of the octagon to its proper position.

Note that the *height* can be set to zero. However, an element of zero *height* and *medium* other than metal will be ignored. An element of zero *height* and metal *medium* models, by default, a thin metal patch. However, starting with version 2.0, this default can be changed so that the element will model a grid of wires located in the *XY* plane (see [Section E 2.2.7](#) for details).

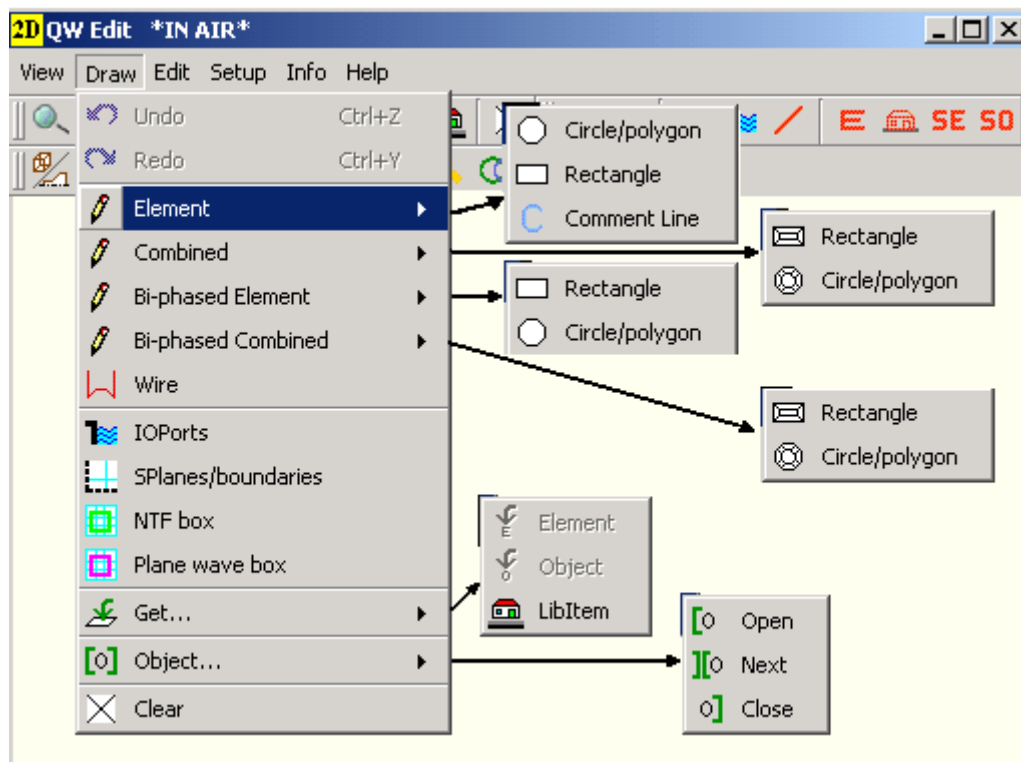


Fig.E 2.2.3-1 Draw group of commands of QW-Edit menu.

- *Combined Element*. This *element* is being introduced at first in a similar way as a normal *element*. However after it is introduced its *bottom* and *cover* can be selected separately, and *cover* can be edited without influencing *bottom*. To check this please draw a rectangular *combined element*. Then please invoke *Edit-Select Element* command and click over the line in which the *cover* part is listed. Now click over the introduced *rectangle*, move it and click again. You can see in the 3D windows the modified shape of the *combined element*.
- *Bi-phased Element*- if this is chosen from the *Draw* menu, you will obtain a *biphased element*.
- *Bi-phased Combined* - if this is chosen from the *Draw* menu, you will obtain a *combined biphased element*.
- *Wire*. This command permits to define thin wires (of radius less than a cell size) parallel to any of the coordinate axes. Starting with version 2.0, the *Wire* command can also be used to define grids of wires in *XZ* or *YZ* plane. We shall consider this possibility in [Section E 2.2.7](#), and herein concentrate on a single *wire*. Modification of the *wire diameter* can be accomplished through the *Edit* command executed on the *wire* (also described in [Section E 2.2.7](#)). Remember that *wire diameter* should be set in current *project units* (in QW-3D versions earlier than 1.8 the *wire*

diameter always had to be set in millimetres). The way of drawing a *wire* is different for Z-oriented wires and for wires located in the XY-plane. To draw an X- or Y-oriented wire, first invoke the *Setup-Level* command and specify the *level* of this wire, setting its *height* to zero. Then invoke *Draw-Wire*, which will allow you to draw the wire as a line segment. Drawing a Z-oriented wire is slightly more complicated. First invoke the *Setup-Level* command and specify the *level* of wire base and the wire *height*. Then invoke *Draw-Wire*, which will allow you (as for the X- or Y-oriented wires) draw a segment of the line in the XY-plane. Do this in such a way that one end of this line is at the desired position of the wire. Then call *Select-Point* and move the other end to the same position.

- *IOPorts*. This command will be described in detail in [Section E 2.4.1](#).
- *SPlanes/boundaries*. This command will be described in detail in [Section E 2.4.2](#).
- *NTF box*. This command is used to create a virtual box needed for antenna simulations. It introduces absorbing boundaries around and sets the boundaries on which the Near-To-Far transformation will be executed for the purpose of calculating the radiation patterns. To introduce a box we draw a rectangle. The box will be created at the current level, with current height (set in Set-Level dialogue) and with the size in XY plane corresponding to the drawn rectangle. The NTF boundary will be separated from the absorbing walls by 10 % of the size of the box. After drawing the *NTF box* we can invoke *Select Element* command to see on the list seven new elements: six absorbing sides and the NTF box. Each of these elements can be edited or deleted like other elements. In particular, we can change the size and position of the *NTF boundary* through the *Edit special, boundary,...* dialogue described in [Section E 2.4.3](#). Above we described a possibility of introducing the *NTF box* in a manual way. However with the new system of libraries introduced in version 2.1 we expect that most of the users will prefer to use the special *NTF box* generating UDOs available in the library `..lelib\boxes`.
- *Plane wave box* creates an auxiliary surface used for exciting an incident plane wave. To introduce a *plane wave box* we draw a rectangle. The box will be created at the current *level*, with current *height* (set in *Set-Level* dialogue) and with the size in XY plane corresponding to the drawn rectangle. After drawing the *plane wave box* we can invoke *Select-Element* to see it on the list, and to select it for editing its size and position (through the *Edit special, boundary,...* dialogue described in [Section E 2.4.3](#)). Parameters of the plane wave to be excited will be defined through the *I/O Ports Parameters dialogue* described in [Section E 2.5.1](#). Similarly to the *NTF box* case we suggest alternative and more convenient way of introducing the *plane wave box* – by using special *plane wave box* generating UDOs available in the library `..lelib\boxes`.
- *Get....* This command is used to read from disk previously saved *elements* or *objects*.
 - *Element* It is not operational in version 6.5.
 - *Object* It is not operational in version 6.5.
 - *LibItem* is used to read an external object from the object library.
- *Object....* This command permits to create *internal objects*. Since an *object* is a composition of *elements* there is sometimes a need to combine some *elements* into an *object* to be able to move them together. All the *elements* drawn between the command *Open* and *Close* (or *Next*) will be assigned to the *object*. The following commands are available:
 - *Open* for opening a new internal object
 - *Next* for closing the current object and opening a next one
 - *Close* for closing the object (the list of elements forming it)

- *Clear*. This command clears entirely the current *project* leaving the space in the *QW-Editor* for a new *project*. However note that the new *project* inherits the environment of the old *project* (number of windows, their size etc). That is why when we want to prepare a new *project*, it is convenient to open an old one dealing with a structure of a similar shape, execute *Draw-Clear* and change the name using *File-Save as*.

E 2.2.4. The *View* group of commands



Fig.E 2.2.4-1 *View* group of commands of *QW-Edit* menu.

The *View* group contains the following commands:

- *Zoom* activates a window zoom. The cursor changes from  to  and now allows marking a rectangular region to be shown, occupying the window.
- *Extents* restores the full picture.
- *Options* opens *2D view options* dialogue with four groups and five buttons:

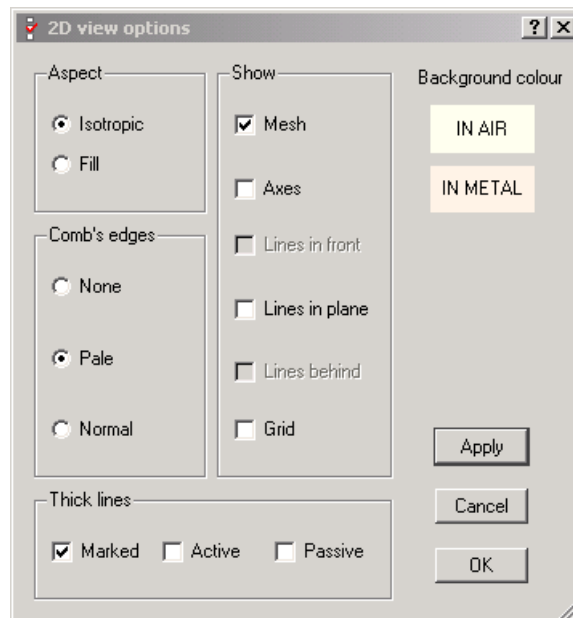


Fig.E 2.2.4-2 *2D view options* dialogue window.

- *Aspect* group allows selecting between *isotropic* display (with natural vertical to horizontal proportions) or *fill* display (filling the entire window with proportions changing with the window proportions).

- *Comb's edges* allows choosing whether the edges of *combined elements* be displayed in *Normal* way, as *Pale* lines, or be hidden (*None*). The latter two options help obtaining a clear picture in *projects* constructed with many *combined elements* modelling geometry variation along the *z*-axis.
- *Show* controls which lines are displayed in the window. *Mesh*, *Axes*, and *Grid* correspond to the *Visible* fields in *Setup-Mesh* and *Setup-Axes*, and to the *Grid* box in *Setup-Grid/Snap*, respectively. When *Show Lines in plane* is selected, only the elements having *bottom* at the current *level* will be displayed.
- *Thick lines* allows choosing whether *Marked*, *Active* and *Passive elements* be displayed with standard or thicker contours.
- *Apply*, *Cancel* and *OK* buttons have standard Windows meaning.
- *Background colour* allows customising the background colour for the circuits defined *in air* and *in metal*, separately.

E 2.2.5. The *Info* and *Help* groups of commands

The *Info* group contains the following commands:

- *Mesh*. *Mesh* refers to the set of FDTD cells. These cells of cubicoidal (or cubicoidal cut by a plane) shape have the basic size defined through the command *Setup-Mesh*. However, the actual cell size may be modified downwards by the software to fill the area between *mesh snapping planes* with an integer number of cells. *Mesh snapping planes* are either introduced explicitly by the user (via *Draw-SPlanes/boundaries* command) or forced automatically by the *QW-Editor* (for example, at horizontal boundaries of *metal elements*). The *Info-Mesh* command permits to see the list of all *mesh snapping planes* (on the left of the box of [Fig.E 2.2.1-10](#)) and the minimum distance between the neighbouring planes (on the right of the box). This permits to spot the planes very close to one another. Such planes would produce very small FDTD cells. Since the step in time of the FDTD analysis must be adjusted to the smallest cell (Courant stability criterion), existence of very small cells can drastically slow down the simulation process. Thus small cells should be avoided whenever not absolutely necessary for correct space approximation of the analysed structure. Since during the editing process the user can unintentionally introduce very small cells, we recommend as a good habit consulting the list of *mesh snapping planes* any time after important changes in the structure have been made. Note that the dialogue describing the special planes can also be reached from the *Setup-Mesh-Splanes info* path. Some additional information has been given in [Section E 2.2.1](#).
- The *Help-Info* command is not operational in version 6.5.

E 2.2.6. Toolbar of *2D window*



Fig.E 2.2.6-1 The toolbar of a *2D window*.

Most frequently used commands of the *2D window* can be invoked using the toolbar icons. The consecutive (from left to right) icons have the following meaning:

	View-Zoom		Edit-Object
	View-Extens		Edit-Select Element
	View-Options		Edit-Select Object
	Draw-Element-Rectangle		Setup-Circuit type
	Draw-Element-Circle/polygon		Setup-Plane
	Draw-IOPort		Setup-Level
	Draw-SPlanes/boundaries		Setup-Axes
	Draw-Get-LibItem		Setup-Grid/Snap
	Draw-Clear		Setup-Mesh
	Edit-Undo (Ctrl+Z)		Info-Mesh
	Edit-Redo (Ctrl+Y)		Edit-Reproduce
	Edit-Point		Edit-Join-Cut
	Edit-Port		Edit-Join-Glue
	Edit-Line		Edit-Join-Intersect
	Edit-Element		

E 2.2.7. Additional remarks on wire grids (asymptotic boundary conditions)

Asymptotic boundary conditions (ASBC) have been introduced following a suggestion of some of our users. ASBC are defined by setting to zero one of the electric field components over the region covered by ASBC, and thus can be realised as *wire grids*. Regarding the orientation and geometry of ASBC, the following restrictions apply in QW-3D:

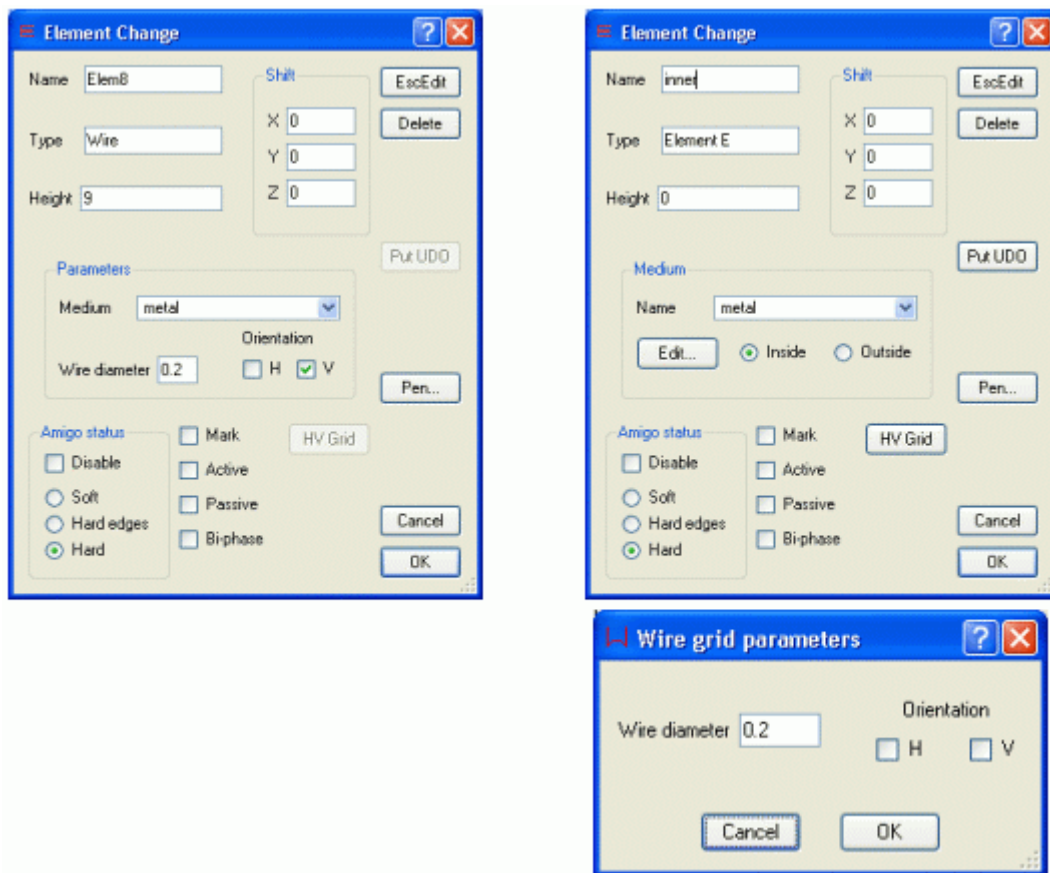
- ASBC can only be defined in planes perpendicular to one of the coordinate axes;
- in vertical planes (parallel to XZ , YZ) ASBC must have a rectangular shape;
- in horizontal planes (parallel to XY) arbitrarily-shaped ASBC can be drawn, but a stair-case approximation will be made.

Note that formally speaking, ASBC can also be introduced by drawing appropriate grids of individual wires. There is, however, one major practical difficulty in such an approach: the user has to draw each wire separately, exactly following the existing FDTD mesh. After re-meshing of the *project*, the wire grid has to be re-drawn. Those problems are avoided by application of special elements, which we shall call “*wire grids*”. Drawing a *wire grid* really means drawing a frame (rectangular in XZ , YZ

planes; arbitrarily-shaped in XY plane), which is then filled with wires of a horizontal (H) or vertical (V) orientation. The number and locations of *wires* in the *grid* automatically adjusts to the mesh and mesh changes. Their *orientation* is chosen by the user in *Element Change* dialogues of *QW-Editor*. [Tab.E 2.2.7-1](#) specifies the correspondence between the horizontal/vertical (H/V) terminology of *QW-Editor*, and the electric field component effectively set to zero.

Wire grids can be defined through the *wiregrid.udo* file, prepared by *QWED* and delivered together with *QW-3D* software. Manual definition of *wire grids* is also possible although less convenient.

ASBC parallel to plane:	ASBC orientation:	ASBC definition:
xy	horizontal (h)	$E_x=0$;
xy	vertical (v)	$E_y=0$;
xz	horizontal (h)	$E_x=0$;
xz	vertical (v)	$E_z=0$;
yz	horizontal (h)	$E_y=0$;
yz	vertical (v)	$E_z=0$;

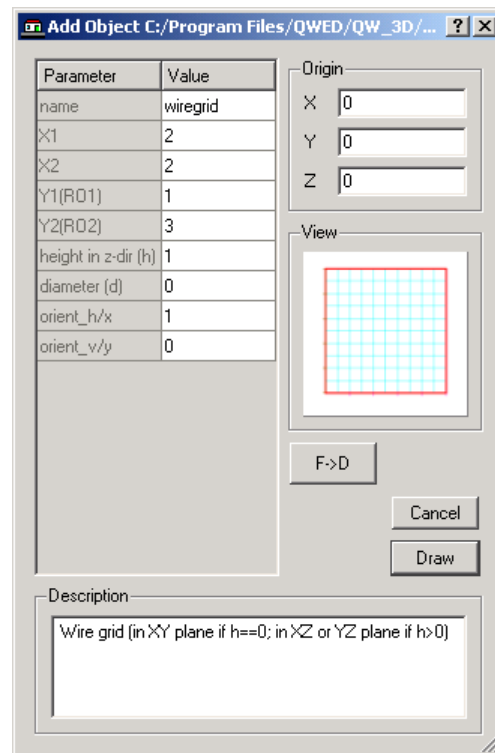


Tab.E 2.2.7-1 Definition of *horizontal* and *vertical* wires in *QW-3D* software. Upper table shows the meaning of *horizontal* and *vertical* depending on the plane of ASBC. The *Element Change* dialogues apply and to a wire element (left) and to a metal element of zero height (right) after *HV Grid* button is pressed.

Definition of ASBC with *wiregrid.udo*

Rectangular *wire grids* can be defined in *QW-3D* with *wiregrid.udo*.

Invoking the *Draw-Get-LibItem* command and selecting *wiregrid.udo* shows the following dialogue with default parameters:



Pressing *Draw* creates a rectangular *wire grid* perpendicular to the *X*-axis, at $x=X1=X2=2$, based at level $Z=0$, with height $h=1$, between $Y1=1$ and $Y2=3$ (all in current *project units*). The grid will be filled with horizontal (*Y* -oriented – see [Tab. E 2.2.7-1](#)) wires. If the FDTD mesh lines are at $z=0,1,2,\dots$, the *y*-oriented wires will appear at $z=0,1$.

The user can modify the parameters, creating different *wire grids*. Note that:

- o The grids must be planar (not volumetric). Consequently, we must make one of the three settings: $h=0$, or $X1=X2$, or $Y1=Y2$. Otherwise a warning will appear.
- o *Orientation* boxes allow to choose horizontal and/or vertical wires. Setting *Orient_h/x*=1 will create horizontal wires, *Orient_v/y*=1 will create vertical wires. If both *orientations* are set to 1, a 2D mesh of wires will be created - this will no longer imitate ASBC, but rather to a thin metal plate. If none of the *orientations* is checked, no *wires* will be created.
- o If $d>0$, all wires in the grid will have diameter d . In such a case, besides setting the tangential *E*-field components to zero, singularity corrections will be applied to the *H*-fields encircling the wire, and *E*-fields perpendicular to the wire.
- o The *X* and *Y* origin coordinates are irrelevant.

Manual definition of XZ and YZ wire grids

A grid of wires parallel to the *XZ* plane can be created by drawing an *X* -oriented wire, in a standard way (see [Section E 2.2.3](#)), but of non-zero *height*. Note that:

- o Default *orientation* of wires in the grid is *horizontal* (*x*-oriented). The uppermost wire will not be created. For example, if the *wire* element is drawn at level $z=0$, of height $h=5$, and the FDTD mesh is at $z=0,1,2,3,4,5,\dots$ – the individual wires will be set only at $z=1,2,3,4$.
- o *Wire orientation* can be changed to *vertical* (*Z* -oriented) in the *Element Change* dialogue, which is shown below [Tab. E 2.2.7-1](#) and can be invoked by selecting the *wire element* on the *Select Element* list, double-clicking with left mouse button, and clicking right mouse button). The *vertical wires* will be of user-defined height h . However, the last *wire* on the right will not

be created. For example, if the *wire element* is drawn between $x_1=1$ and $x_2=3$, and the FDTD mesh lines are at $x=0,1,2,3,\dots$ – the individual *vertical wires* will be set only at $x=1,2$.

Analogous rules apply to wire grids parallel to the *YZ* plane (*Y* replacing *X* in the above description).

Manual definition of XY wire grids

A grid of wires parallel to the *XY* plane can be created by drawing a metal *element* of zero *height*, of an arbitrary shape, in a standard way (see [Section E 2.2.3](#)). The *Element Change* dialogue for such *elements* has now been supplemented with *Advanced* parameters part, and is shown below [Tab. E 2.2.7-1](#). It allows to choose between a metal patch or a *wire grid*, and to specify *orientation* and *diameter* of wires. Note that:

- o If only *H* or *V orientation* box is checked, the *X*- or *Y*-oriented *wires* are created, respectively. The last *wire* in the positive *X* – or *Y*-direction, respectively, is not created.
- o If both *H* and *V orientation* boxes are checked, a 2D grid of the *X*- and *Y*-oriented *wires* is created. Stair-case approximation of the shape is applied. The last *wires* in the positive *X*- and *Y*-directions are not created.
- o If none of the *orientation* boxes is checked (default), a thin metal patch is conformal modelled, as in the previous versions of the software.

E 2.3. The 3D windows

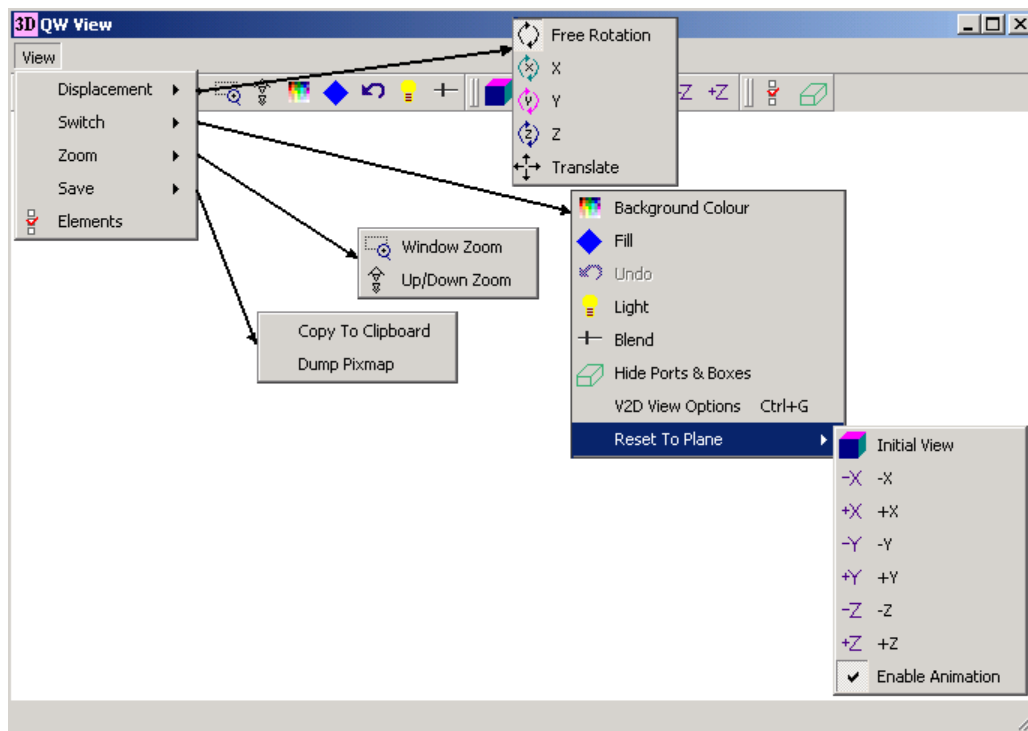


Fig.E 2.3-1 View group of commands of 3D QW View menu.

The 3D windows are of QW View type. They allow *project* visualisation but do not allow any kind of *project* editing. In the 3D window we have one group of commands:

- *View*. This group contains the commands:

- o *Displacement*: after pressing left mouse button over the window, the cursor changes to  and its movement over the window causes displacement of the image. The type of displacement can be selected here as:
 - ◆  *Free Rotation*
 - ◆  *X Rotation*
 - ◆  *Y Rotation*
 - ◆  *Z Rotation*
 - ◆  *Translate*
- o *Switch*:
 - ◆  *Background Colour* opens a standard palette for selecting the background colour
 - ◆  *Fill* switches between wire and solid displays
 - ◆  *Undo* undoes the last displacement operation
 - ◆  *Light* pressed adds light to the image
 - ◆  *Blend* blends the applied colours with white, which produces translucency effects for solid displays
 - ◆  *Hide Ports & Boxes* pressed hides all non-geometrical entities of the *project* (I/O Ports, contours, FD Monitors, NTF, PML, Subgrid, Plane Wave, Mur etc.)
 - ◆ *Reset to Plane* contains shortcut commands for setting the direction from which the image is seen:  *Initial View*,  *from -X*,  *from +X*,  *from -Y*,  *from +Y*,  *from -Z*,  *from +Z*
- o *Zoom*: a convenient way of zooming the image is with a mouse wheel. Additionally, there are two options:
 - ◆  *Window Zoom* pressed causes that after pressing left mouse button over the window, the cursor remains  and allows panning a region of the window.
 - ◆  *Up/Down Zoom* pressed causes that after pressing left mouse button over the window, the cursor changes to  and its movement up or down zooms and unzooms the image, respectively. .
- o *Save*:
 - ◆ *Copy to Clipboard* captures window contents to clipboard
 - ◆ *Dump Pixmap* saves window contents to a **bmp* file, of user-defined name.
- o  *Elements* opens *View Elements* dialogue, which allows showing or hiding individual elements. Its form for the *wgtocx1.pro* example is shown in [Fig.E 2.3-2](#). *Elements* can be selected / unselected with mouse clicks or via *Sel.All* / *Unsel.All* buttons. *Elements* can be shown or hidden by pressing *Show Sel* (or *Show All*) or *Hide Sel* (or *Hide All*), confirmed by *Apply* or *OK*. Double-click over *H* or *V* letter in *Visible* column, followed by *Apply* or *OK*, is

an alternative way to show or hide a particular *element*. All other columns are for information only. Show and hide operations performed from this dialogue prevail over possible previous *Hide Boxes* command. *Cancel* exits the dialogue without performing the last requested unconfirmed operations.

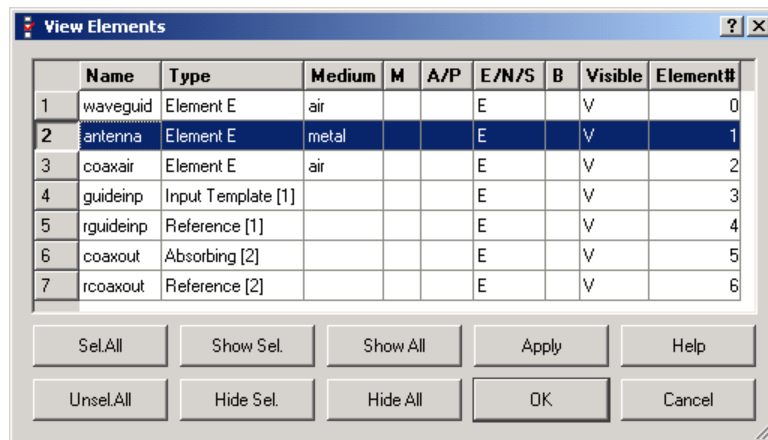


Fig.E 2.3-2 Example *View Elements* dialogue for the *wgtoctx1.pro* example with *antenna* element selected.

The colour assigned to a particular *element* is used for its display in all windows. It can be changed in *2D QW Edit window* of *QW-Editor* only:

- ♦ move to a *2D QW Edit window*,
- ♦ invoke *Edit-Select Element* command and choose the element in question,
- ♦ press the right mouse button to get the *Change element* menu,
- ♦ press *Pen* to enter *Select Pen* dialogue, and choose a new *Colour* (note that *Style* and *Width* chosen in the same dialogue apply to *2D window* only),
- ♦ confirm new settings with *OK* buttons.

An alternative to *3D window* solid view display resides in using *HOOPS 3D Viewer*. The *Tools-SAT Viewer* command opens the viewer, transforms the current *project* description into a format recognised by the viewer, and loads it. We can then use the rich menu of the *HOOPS 3D Viewer* options to display and rotate the 3D image of the designed structure.

E 2.4. Drawing and editing ports

E 2.4.1. Drawing I/O Ports and reference planes

As it has been shown in [Chapter UG 2](#) or [V2D 2](#) of the manual in most practical application we can use predefined library UDOs to draw appropriate ports. This section describes the procedure for drawing ports in an alternative - manual way. In fact the commands, which can be found in, the scripts of the port generating UDOs fully correspond to the manual procedures described below. After invoking the command *Draw-I/O Ports* we get a dialogue window presented in the upper part of [Fig.E 2.4.1-1](#).

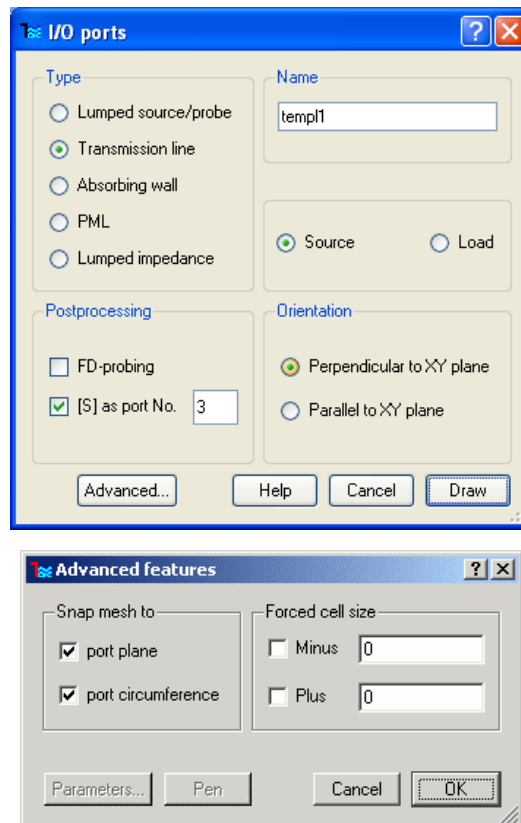


Fig.E 2.4.1-1 Dialogue windows for drawing I/O ports.

On the left hand side of the window we must define the *type* of the *port*. There are four exclusive possibilities:

- *Lumped source/probe* describes a port defined by its current and voltage at certain nodes. In version 6.5 we can consider lumped ports only across one FDTD cell. The point position will be fixed by a left mouse button click or by entering the coordinates from the keyboard (after pressing **K**-key). Orientation is irrelevant. The circuit model of the lumped port consists of a voltage source in series with a resistor. It reduces to:
 - ♦ a resistor if the source is deactivated,
 - ♦ an ideal voltage source if resistance is set to zero,
 - ♦ an ideal current source if resistance is set to +INF.

Regarding sources, note that they become electric voltage/current sources if the port is connected to one of the electric field components, and magnetic voltage/current sources if the port is connected to one of the magnetic field components.

The lumped port will serve as a source or a probe depending whether *Source* or *Load* will be checked in the *I/O ports* dialogue. In both cases, in the *I/O Ports Parameters* dialogue (see also [Section UG 2.11.2](#), STEP 7) the user must define:

- ♦ a field component to which the port is connected,
- ♦ internal resistance of the port (a value from the 0..+INF range, or *self adjusted* to match the neighbouring FDTD mesh).

For ports checked as *Source*, the user will additionally define the source waveform, amplitude and delay. Setting *no excitation waveform* effectively transforms the *source* into a load. For ports checked as *Load*, zero exciting amplitude is assumed and the lumped port acts as a lumped resistor.

The *Lumped source/probe* port can be used in the following cases:

- ♦ as a virtual source in eigenvalue problems (resonant modes, their frequencies and Q-factors in a resonator),
- ♦ as a dipole (in the analysis of dipole-excited antenna problems),
- ♦ as a lumped source of finite output impedance (in the calculations of embedding impedance for lumped elements),
- ♦ as a probe (especially in millimetre-wave problems - taking part in S-parameter extraction together with *Transmission line* ports).

In the latter two cases, the point source or probe should be placed in a one-cell gap between two metal elements. If the gap (for example between two metal plates) is larger than one cell, the source or probe should be connected to the metal elements with additional wire elements.

It is also possible to place a point source or probe directly on the wire. This feature will be particularly convenient in the analysis of wire antennas: changes in the meshing will not need to be associated with changing the size of the gap, to keep it equal to one FDTD cell. The *Lumped source/probe* port placed on the wire will “cut” a gap in the wire and work directly in that gap.

If a *lumped source/probe* is included in the S-parameter postprocessing, its resistance R serves as a reference resistance for separating the incident and reflected waves. However, if $R=+\text{INF}$ or $R=0$, then $50\ \Omega$ is taken as a reference.

Note that *FD-Probing* postprocessing also produces reflection loss as one of its results. It is denoted by a symbol S_number , where “number” is the value of R . Here S_number is not calculated if $R=+\text{INF}$ or $R=0$.

Postprocessing part of the dialogue allows specifying in which postprocessing algorithms the *Lumped source/probe* port should take part:

- ♦ *FD-probing* stands for frequency domain probing, and means that Fourier transforms of terminal voltage and current will be calculated,
- ♦ $[S]$ as *port No.* includes the *Lumped* port in the S-parameter extraction system together with *Transmission line* ports.
- *Transmission line* describes a port defined by integrals of fields over a cross-section of an arbitrary transmission line. In such a case we are always interested in a particular mode of propagation in the line. To extract any parameters at the *Transmission line* port we will need to calculate a mode *template* (as explained in further Sections). Before clicking *Draw*, the user must specify whether the port is *Perpendicular* or *Parallel to XY plane* (which entails that it will be drawn as a line or rectangle, respectively).
 - ♦ The port can be defined as *Source* or *Load*. The declaration of the port as a *source* means that it will serve as the input port in the case of *Sk1* element analysis, which uses excitation from one port only. To avoid confusion in the S-matrix interpretation it is recommended that this port be marked as port number one. In the case of *S_{mn}* analysis each port included in S-parameter extraction will serve as a source in one of the consecutive simulations.
 - ♦ If the *Transmission line* port is included in S-parameter extraction (via $[S]$ as *port No.* option), a *reference plane* will be automatically generated by *QW-Editor* at a default distance from the port. If the $[S]$ option is not checked, the port can serve as *source* or *load* but it will be ignored from the point of view of S-matrix analysis.

- *Absorbing wall*. Such a port is used for absorbing termination of a chosen rectangular part of the FDTD mesh. It is not used for extracting any parameters for further postprocessing. The absorbing boundary condition is *Mur with superabsorbing*.
- *PML*. Such a port is used and works like *absorbing wall*, but in this case the absorbing boundary condition is Perfectly Matched Layer.
- *Lumped impedance* option indicates a place where a lumped impedance is inserted across one FDTD cell. The choice of the type of that impedance and its numerical parameters are explained in [Section E 2.5.1.3](#).

Note that:

- The user can mark several *I/O ports* as *sources*. In this case:
 - If no *S-parameter postprocessing* is chosen, all these *ports* will be simultaneously excited with waveforms assigned to them in the *I/O Ports Parameters* dialogue. In particular, the waveforms may have different *amplitude* and *delay* values.
 - If *Smn* postprocessing is chosen, each port will serve as a source in a different simulation, with other ones serving as loads.
 - If *Sk1* postprocessing is chosen, all ports will act with waveforms assigned to them and port No.1 will be treated as a reference port. The results of such a postprocessing will be inconsistent with standard understanding of S-matrices, but may be useful in some unusual studies.
- The *sources* and *reference planes* are marked by arrows corresponding to the anticipated direction of the energy flow (see [Fig.E 2.4.1-2](#)). The *absorbing ports* are marked by triangles with vertices towards the direction from which the wave is coming (imitating unechoic chamber walls). The orientation of the *I/O ports* and *reference planes* is important. After drawing them please verify if the orientation is correct, and if necessary change it using the *Edit-Port* command.
- In the upper right part of the *I/O ports* dialogue we should define the name of the port. This name may be useful for example to select a particular *port* on the list of *elements*. We can also change the *number* assigned to the *port* for the purpose of the S-parameter calculations.
- The ports which take part in S-matrix calculations must bear consecutive numbers. For example assigning to three available ports the numbers 1, 2 and 4 will result in computation error.
- We should avoid the situation in which the FDTD cells adjacent to two sides of a *reference plane* differ significantly in size measured along the direction of propagation. This introduces some error into S-matrix calculation.

[Fig.E 2.4.1-2](#) shows a zoomed display of a transmission line input port. It emphasises one more crucial issue in the *transmission line port* definition. Namely, line marking the port should be drawn parallel to one of the coordinate axis, precisely within the metal body of the waveguide. In fact, *QW-Editor* will not allow you to draw the *port* obliquely. However, if you draw the *port* parallel to the *Y*-axis, but either too short or not starting precisely at the metal wall, only a part of the guide cross-section will be used for energy injection, leading to wrong simulation results. If you draw it too long, protruding into the metal wall of the guide or even further, the simulation results may be correct, but the simulation time may drastically increase if the mesh snapping properties of the *port* circumference imposed on the mesh snapping by the geometry produce a very small cell. This discussion underlines that if the user decides to draw a port in a manual way this operation needs to be done with special care.

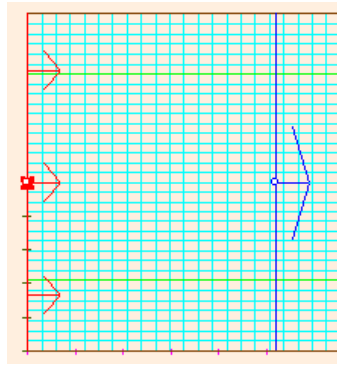


Fig.E 2.4.1-2 A fragment of a 2D *window* showing input port (marked by a red line with three red arrows) and its reference plane for S-parameters calculation (marked by a blue line with one blue arrow).

The ports in XZ and YZ planes are introduced in a way similar to *elements* with the only difference that their base is a straight line while the *elements* have polygonal bases. The ports in XY planes are introduced in a similar way as a rectangular base of an *element*.

Note that checking the port for $[S]$ automatically introduces a *reference plane* situated at some distance from the *port* ([Fig.E 2.4.1-2](#)). The *reference plane* is the plane at which the data for S-parameter calculation is collected. The user should move it to the proper position taking into account the following criteria:

- It must be situated in a place where the cross section of the I/O transmission line is the same as in the I/O *port* plane.
- It must not be closer than two cells from the I/O *port* but for better accuracy it should rather be moved at least 4-5 cells towards the circuit.
- The S-parameters are calculated using the vector product of the simulated fields and the mode *templates* to filter out the influence of the modes other than the considered one. Nevertheless the presence of other modes may have some influence on the accuracy, especially when calculating wide-band characteristics of inhomogeneously filled lines. That is why it is recommended that the *reference plane* be kept at a safe distance from the discontinuities, which produce a high, content of unwanted modes.
- The position of the *reference plane* rectangle in the plane perpendicular to the direction of propagation should be exactly the same as of the corresponding *port*. If you move the *reference plane* graphically on the screen please observe that it is moved only in the direction perpendicular to the port plane (use the *Zoom* function to verify this). However the best way to move a *reference plane* is described below. Invoke *Select Element* command and select the *reference plane* in question. Then press the right mouse button to get into the *Edit special planes* dialogue window. In this plane use *Set plane* command or *Shift* command to put the reference plane to the desired position.

Let us also note that it is possible to define in the same place two different *ports* operating on two different transmission line modes (like for example vertical and horizontal polarisation in a square waveguide). In this case we define in the same place two I/O *ports* and assign to each of them different mode *template* by a proper choice of excitation in *template generation*. Such a process has been exemplified in [Sections UG 2.2.3-UG 2.2.5](#).

Further choices regarding each I/O *port*, and in particular its excitation and template generation, are made in the I/O *Ports Parameters* dialogue invoked via *Parameters-I/O Ports* command. We will discuss these choices in the [Section E 2.5](#).

E 2.4.2. Drawing mesh snapping planes and symmetry planes

After invoking the command *Draw-SPlanes/boundaries* we obtain the dialogue of [Fig.E 2.4.2-1](#). The most frequently used application of this command will be to *force mesh* through the introduction of a *mesh snapping plane*. It is like indicating a plane to which the FDTD mesh must adjust:

- *electric* forces cell edges i.e., a plane whereat the tangential electric field components are calculated,
- *magnetic* forces cell centres i.e., a plane whereat the tangential magnetic field components are calculated,
- *neutral* forces either *magnetic* or *electric* plane, depending which distorts the existing mesh to a lesser extent,
- *weak* forces an additional plane between the FDTD mesh planes whereat additional geometry information will be extracted for better approximation of complicated shapes at subcellular levels; in AMIGO operation it may force *electric* plane if this within mesh constraints set by the user.

Technically mesh snapping planes work like a port without executing the input/output functions. Note that in the case of *Mesh snapping planes* the size of the rectangle introduced to mark the plane is of no importance. It is displayed only to recall to the user where the plane is. The user should make it of the size least disturbing to the images of the structure or even choose the white colour to make it invisible. A keyboard action SHIFT+M (or SHIFT+m) moves the *x*- and *y*-mesh snapping planes to the borders of the *project* and normalises their lengths of the displayed markers to a standard cell sizes.

The right part of the dialogue allows fast inserting of *electric mesh snapping planes*. Only *orientation* needs to be selected and one coordinate (*position*) needs to be specified, and *QW-Editor* will draw the plane with default dimensions and colour.

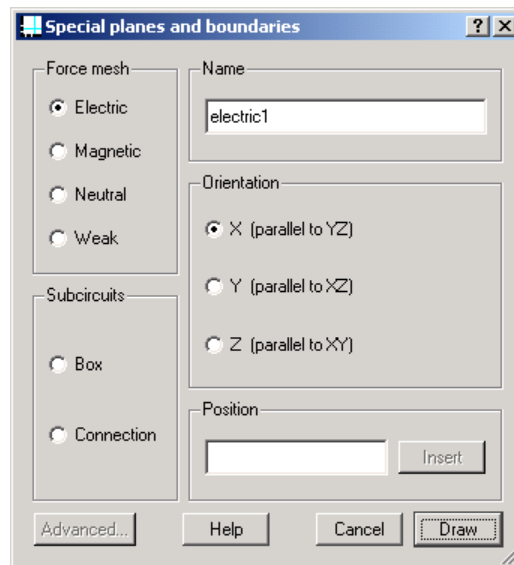


Fig.E 2.4.2-1 A dialogue for drawing special planes and boundaries.

Another group of application of the *special planes* is in defining and connecting the subcircuits of the *QW-Simulator* or connecting the *QW-Simulator* to external simulating tools.

E 2.4.3. Editing ports

Once a *port* (in a general sense) has been defined, it may be necessary to modify its position. This can be done using *Edit I/O port* dialogue (which appears for *I/O ports*) or *Edit special, boundary...* dialogue (which appears for *reference planes*, *mesh snapping planes*, *symmetry planes*, *NTF boxes* and *plane wave boxes*), shown in [Fig.E 2.4.3-1](#). In this window we can change several previously defined parameters of the *port*. Let us point out only some of the specific commands available here.

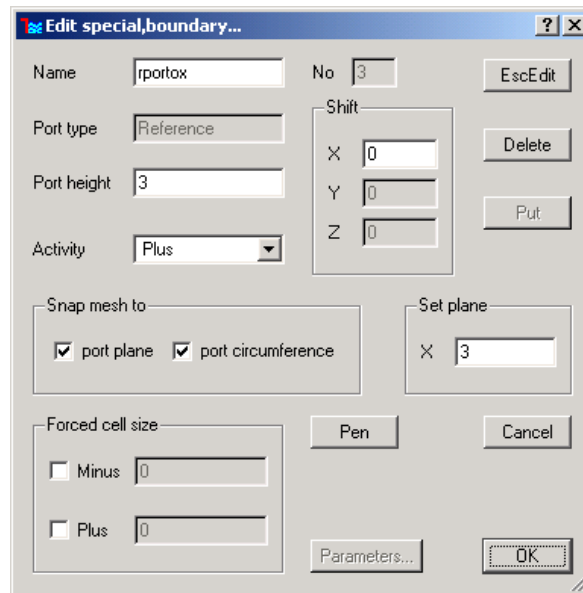


Fig.E 2.4.3-1 A dialogue for editing ports.

- *Activity*. This command allows changing orientation of the port. For example it is natural that a port defined at the left edge of the structure will assume the circuit to be on the right and the software will make such a default assumption. However if we choose a port inside the structure an ambiguity appears. In this case the default choice can be corrected here by the user.
- *Set Plane*. Here we can verify the position of the *port* in the direction perpendicular to it. We can set another position. The advantage with respect to doing this with a mouse in *2D window* is that we can be sure that the position of the *port* in the cross-section of the line has not been changed.
- *Shift* allows to move the port by a certain distance in the direction of each of the three axes. Note that in the case of *reference planes*, *shift* will only be allowed along the transmission line.

E 2.5. Parameters of wave simulation

E 2.5.1. Input/Output Port parameters

In the *QW-Editor* we do not only prepare the shape of the considered structure and define the FDTD mesh but we also set several important parameters of the FDTD simulation. Among them are the *I/O Ports Parameters*.

The *I/O Ports Parameters* dialogue box can be accessed by the path *Parameters-I/O Ports*. The *I/O Ports Parameters* dialogue changes, depending on the type of port. In the next five subsections we will discuss the shape of the dialogue for different types applicable in *QW-3D* software. In subsection [E 2.5.1.5](#) we will concentrate on *QW-V2D* applications of ports.

E 2.5.1.1. I/O Port Parameters in the case of transmission line ports

Transmission line ports have the following features:

- Each of them is characterised by its geometry, as described by the user in the shape editor, and by a particular propagating mode. Port parameters should allow *QW-Simulator* to generate the field distribution of the desired mode in a normalised form. Such a distribution is called the *mode template*. In general, one geometrical port may be represented by several electromagnetic ports - each of them associated with a different *mode template*.
- Transmission line ports are defined for S-parameter extraction. To allow application of the differential method, they must be accompanied by *reference planes* at a distance of minimum 3 FDTD cells from the port.
- The port must be defined on a segment of transmission line such that its geometry and material filling between the port plane and the *reference plane* and three cells beyond the *reference plane* do not change.
- Each port is labelled as a *source* or a *load*. This marks its function when the S-matrix are calculated with excitation from just one port (S_{k1} postprocessing). In the case of S_{mn} postprocessing, *QW-Simulator* performs several runs with consecutive ports acting as sources.

The class of transmission line ports may be divided into four subclasses: *TEM*, *multiTEM*, *planeTEM* and waveguides. We will start with a *TEM* port. [Fig.E 2.5.1.1-1](#) presents the upper (the only relevant) part of the *I/O Ports Parameters* dialogue in the case of *coaxout* port of the *wgtocx1.pro* example discussed in *QW-3D User Guide*. The choice of *TEM* is marked in the *Exciting field* box. Scanning the dialogue from the left we can see that:

- port name is *coaxout*,
- it has been labelled as a *I/O LOAD*,
- it will be referred as *No.2* in S-matrix calculations,
- *Auto* checked means that during the port operation as a *load* *QW-Simulator* will take the effective permittivity (needed to optimise the absorbing boundary condition) from the results of TEM template generation,
- port is not a subject of symmetry condition (either *Horizontal* or *Vertical*),
- if a role of the *source* is assigned to that port during the run with S_{mn} postprocessing, it will be excited by a pulse of spectrum $15 < f < 25$ [GHz], with *duration* of 3 periods of the upper frequency, the *amplitude* equal 1 and *delay* equal 0.

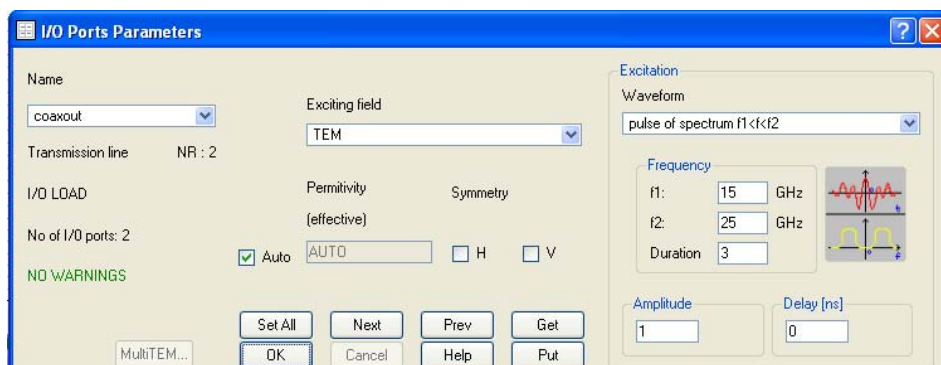


Fig.E 2.5.1.1-1 The upper part *I/O Ports Parameters* dialogue box in the case of a TEM port.

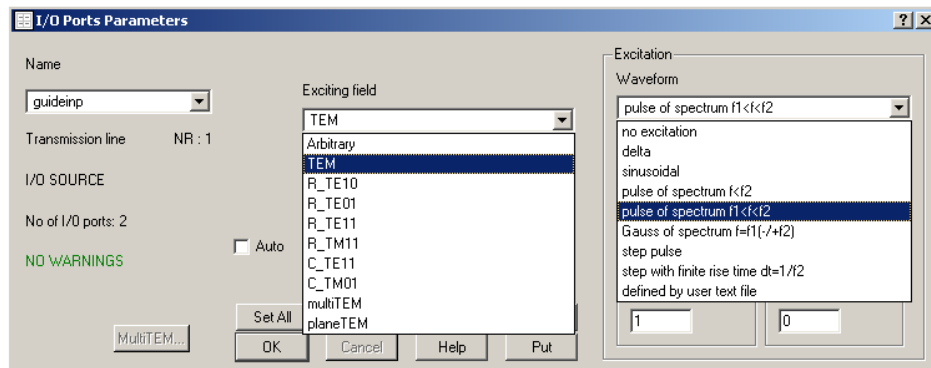


Fig.E 2.5.1.1-2 The upper part *I/O Ports Parameters* dialogue box showing the choice of options of *Exciting field* and *Excitation waveform*.

There are several options for *excitation waveform* as shown in [Fig.E 2.5.1.1-2](#). After choosing one of them its time domain shape and frequency domain spectrum are displayed schematically in the *Excitation* part of the window (see [Fig.E 2.5.1.1-1](#)). The picture is only qualitative. There is however a way of obtaining precise information about a spectrum of the pulse. It is available with *FD-Pavailable* postprocessing described in [Section E 2.5.2](#).

For typical S-parameters calculations we advise the waveforms of limited spectrum $f < f_2$ or $f_1 < f < f_2$, with the limits f_1 and f_2 close to the limits of the frequency band of interest. A pulse with a wider spectrum may excite unwanted resonances outside the band of interest, which prolong the transient effects in the modelled circuit and thus the computing time. In the case of cylindrical waveguide ports, resonances can be expected around cut-off frequencies of the consecutive modes. Beside the frequency spectrum limits, there is a parameter describing the pulse *duration* expressed as the number of periods of the signal of the frequency f_2 . Setting longer pulse duration makes the pulse spectrum closer to a rectangular shape, cutting the unwanted frequencies more effectively, but it also prolongs the computing. The *duration* of the length 3 to 4 is usually a wise compromise between these conflicting requirements.

Sometimes we need a better insight into the wave propagation at a particular frequency. In this case we use the *sinusoidal* waveform. Note that since the sinusoidal excitation can provide the S-parameters only at one frequency, and since the simulation with such an excitation is not faster than with the pulse, it is never used for S-parameter analysis.

In the case of quasi-TEM lines we can often predict that there will be no resonance within the frequency band quite above the band of interest. In such special cases, it may be better to set f_2 higher than the upper frequency of interest since this will reduce the absolute duration of the pulse, and speed-up the convergence.

Two other parameters seen in the considered dialogue window are *amplitude* and *delay* of the signal. They are irrelevant for the S-parameter calculations. They may be important only for visualisation of the effects of concurrent multi-port excitations or for simulation of nonlinear (e.g. temperature dependent) effects.

Note that:

- In case of *template* excitation, there is a direct correspondence between *waveform amplitude* and power available from the source. For *amplitude* set to unity, power available from a sinusoidal source is 1W (at the source frequency) while power available from a delta source - 1W wide-band. For other *waveforms*, available power varies with frequency, and equals unity at the centre of the band. The *FD-Pavailable* postprocessing shows the available power versus frequency. Increasing *amplitude* by a factor of A means increasing the available power by A^2 . For more information

about the normalisation needed to read the absolute values of fields and power please refer to [Section S 2.7](#).

- *Delay* is expressed in [ns]. In the case of sinusoidal excitation, it is directly related to the phase shift, which may be interesting in some applications.

Note also eight buttons used to scroll the list of ports (*Next*, *Previous*), to copy the parameters to all other ports (*Set All*), read the set of parameters to a disc file (*Get*) or write it to a file (*Put*), get help (*Help*) and leave the dialogue (*OK*, *Cancel*).

Let us briefly explain how the TEM port will be generated by *QW-Simulator*:

- In the first phase, the software will search for a conductor. Note that every *I/O port* has a distinguished *point*, which is automatically set in the centre of the *port* but can be moved by the user (with the *Edit-Point* command). If this point is within a conductor, this particular conductor will be selected as a "hot" one. Otherwise the choice of the conductor will not be controlled by the user and a warning will appear in *QW-Simulator*.
- In the second phase, *QW-Simulator* will set the potential of the selected conductor to unity and of all other conductors - to zero. Then it will perform quasistatic finite-difference analysis, generating the electrostatic and magnetostatic fields. Note that *QW-Simulator* will stop with an assert if it finds only one conductor in the line's cross-section. This may happen when the discretization is too coarse and two physically separate conductors merge on the FDTD grid. To avoid merging, short nodes belonging to two conductors must be separated by at least one non-metal node.
- After terminating the TEM template generation *QW-Simulator* will display in the *Simulation log* the characteristic impedance of the considered TEM line and its effective permittivity.

[Fig.E 2.5.1.1-3](#) presents example of the upper part of the *I/O Ports Parameter* dialogue when *multiTEM* option is chosen in the *Exciting field* box, and we have clicked over *MultiTEM* button. This is applicable to multiconductor TEM lines. The example is taken from *QW-3D* [UG 2.2.10](#). In the considered project, we have three different metals. *MultiTEM* option allows defining TEM templates by assigning a specific quasi-static potential to each of the conductor, as specified in the *Conductors potentials* box.

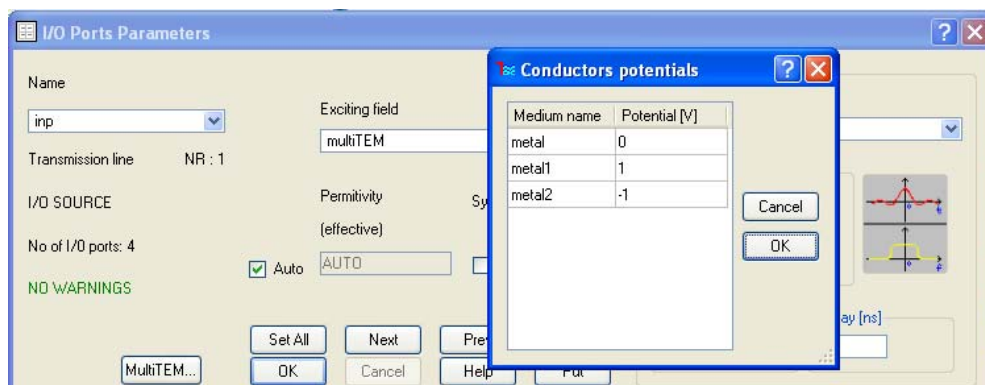


Fig.E 2.5.1.1-3 The upper part *I/O Ports Parameters* dialogue box in the case of a *MultiTEM* port.

There is one more option of the TEM family. It is called *planeTEM*. It is designed for a specific case. When we consider a rectangular TEM port limited by two electric and two magnetic walls, the field distribution is constant in space. Thus we do not need to calculate it but can just assume the constant field distribution with a vertical (V) or horizontal (H) polarisation. We need to be careful when applying *planeTEM* excitation. It is up to the user to verify that the port obeys the above mentioned assumptions and that the polarisation is correctly chosen. Its application may save quite a lot of

computing time when *planeTEM* assumptions are met and the port is large in terms of the number of FDTD cells. .

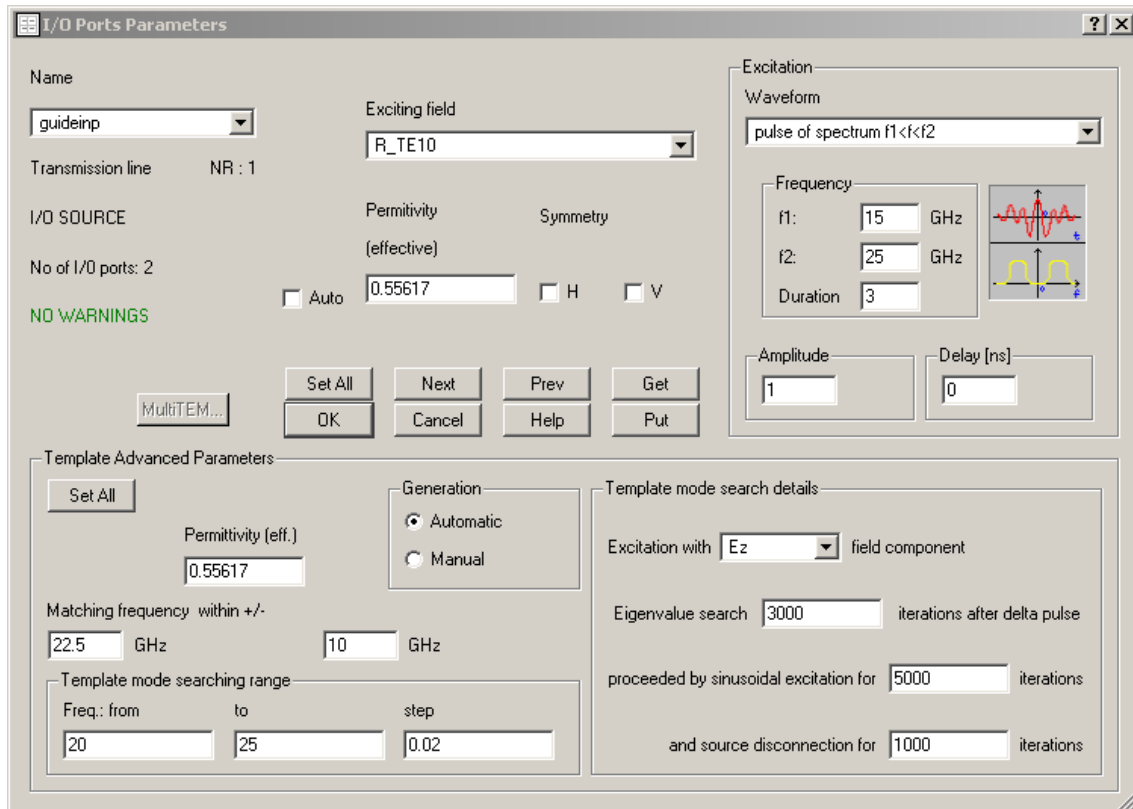


Fig.E 2.5.1.1-4 I/O Ports Parameters dialogue window describing the parameters of dynamic template generation.

Now consider a *dynamic template*. Such a *template* is used basically for waveguide modes. However, it is also applied for quasi-TEM modes in inhomogeneously filled lines at higher frequencies where dispersion effects become important. Dynamic template generation is a subject of detailed discussion in many examples in *QW-3D User Guide*. Refer to sections [UG 2.2.4](#), [UG 2.2.8](#), [UG 2.2.9](#) and [Section UG 2.11.1](#) (STEP 7). The discussion will not be repeated here. We will only formulate some remarks concerning the dialogue, as exemplified in [Fig.E 2.5.1.1-4](#). Note that:

- For the rectangular and circular waveguide modes explicitly listed under *Exciting field*, *QW-Editor* can introduce the default parameters for *QW-3D* analysis. In *QW-V2D* operation, this capability is restricted to the following circular waveguide modes: TM01, TM11, TM21, TM31, TM41, TM02, TM12, TM22, TE01, TE11, TE21, TE31, TE41, TE51, TE02, TE12, TE22. In calculation of the default parameters the software assumes that the size of the port matches the size of the waveguide and that the waveguide is filled with air. If this is not true, the user should introduce the proper corrections to the set of parameters.
- For a general *mn* mode, the index *m* is assumed to correspond to: *x* dimension for ports located in the *XY*-plane or *XZ*-plane; *y*-dimension for the ports located in the *YZ*-plane.
- The *symmetry* boxes *H*, *V* should be checked if only half of the waveguide has been drawn (with respect to *m* or *n* index) utilising symmetry of the circuit. This instructs the software to take double port dimension along *m*, *n* for calculating cutoff frequency, and thus *effective permittivity*, for the *mn* mode.

- *Effective permittivity* is calculated for a previously set *matching frequency*. If the user subsequently changes the *matching frequency*, *effective permittivity* is not automatically recalculated. The simplest way to recalculate it is to re-select the desired mode.
- For other modes, inhomogeneous filling or quasi-TEM lines at high frequencies the user should select the mode to be *Arbitrary* and proceed according to the guidelines formulated in the *User Guide*.
- The software will try to generate template at the *matching frequency*, within a specified margin. Template mode searching range should include this frequency. A warning will be issued if the matching frequency cannot be found.
- *Excitation field component* - this is the component used for mode template generation. It should be set in accordance with the expected template mode distribution as described in [Sections UG 2.2.4-UG 2.2.6](#).
- *Number of iterations for template generation* – We have three numbers here but they are mutually dependent as described in [Section S 2.2](#). A typical value sufficient for obtaining a steady state with sinusoidal excitation in most applications is about 10000. However, the user is advised to observe the field distribution during the sinusoidal stage of the template generating process (see [Section S 2.2](#)) to find out if for his applications this number should be cut down (to save computing time) or increased (to get better accuracy of the final mode *template*).

Summarising the discussion of *arbitrary templates*, let us stress that QW-3D is very flexible in terms of choice of the mode in arbitrarily shaped and inhomogeneous transmission lines forming the ports of the structure. The price for this flexibility is the need to set several parameters enabling the software to find the mode of interest. In general, this is difficult to avoid and the following rules-of-thumb for setting *effective permittivity* may be useful:

- any guide, any mode at cut-off: *effective permittivity* = 0
- empty cylindrical waveguide filled with air:

$$\text{effective permittivity} = 1 - (\text{cut-off frequency of the mode} / \text{operating frequency})^2$$

- cylindrical waveguide filled with dielectric of ϵ_r :

$$\text{effective permittivity} = \epsilon_r \{ 1 - (\text{cut-off frequency of the mode} / \text{operating frequency})^2 \}$$

- cylindrical waveguide filled with material of ϵ_r and μ_r :

$$\text{effective permittivity} = \epsilon_r \mu_r \{ 1 - (\text{cut-off frequency of the mode} / \text{operating frequency})^2 \}$$

- inhomogeneous quasi-TEM line including dielectrics of $\epsilon_1 < \epsilon_2 < \dots < \epsilon_n$:

$$\epsilon_1 < \text{effective permittivity} < \epsilon_n$$

E 2.5.1.2. I/O Port parameters in the case of absorbing walls

There are two types of absorbing boundary conditions (ABC): *Mur with superabsorption* and *PML* (see [Fig.E 2.5.1.2-1](#)). Here are some remarks about their application.

- In general the *PML ABC* is more accurate and its performance is less dependent on the angle of incidence. However, it has some disadvantages:
 - It is rather time and memory consuming
 - It cannot be placed in a region of boundary-conforming cells of shapes other than rectangular

- Practically it is stable only when its sides coincide with the grid termination or a metal wall. Thus it is recommended mostly for antenna problems (when we use the absorbing box surrounding the antenna) or for a reflectionless termination of a rectangular waveguide.
- The user can choose the *PML profile* type (*parabolic* or *exponential*) and the *thickness* of the PML layer (expressed in FDTD cells). The application of those options is discussed in [Section UG 2.3.3](#).
- *Mur ABC* is more flexible. It can be applied practically in any case and its implementation takes into account the conformal cells. Thus it is a standard choice in the case of termination of arbitrarily shaped transmission lines. As shown in various in the *User Guide*, it can also be quite effective in antenna problems. However, since the *Mur ABC* assumes travelling wave conditions, it should be placed at least half wavelength from the antenna to avoid intense interaction with antenna near fields. If the interaction is high, it may cause increased computational errors or even algorithm instability.
- In the case of *Mur ABC* the user sets only one parameter - the effective permittivity of the medium in which the wave propagates. It is used by the software to calculate the correct phase velocity of the wave. Incorrect assumption of the effective permittivity increases the level of spurious reflections from the ABC.

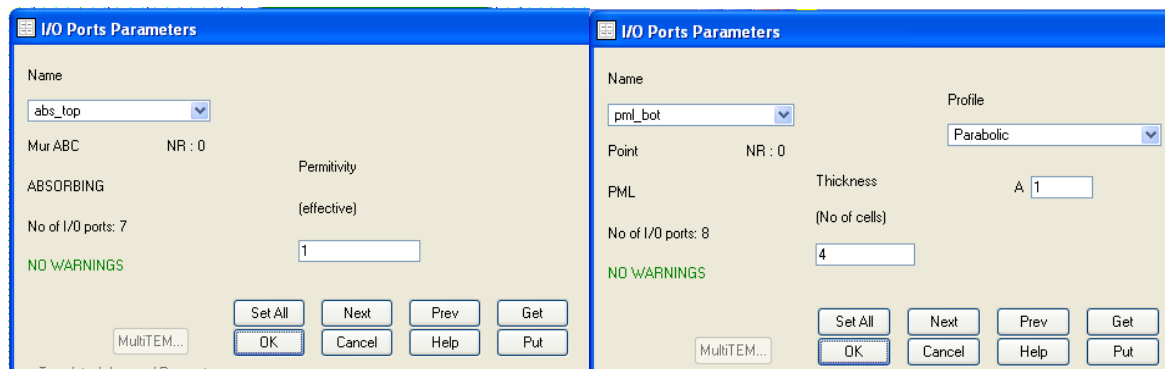


Fig.E 2.5.1.2-1 The relevant parts of the *I/O Ports Parameters* dialogue window in the case of absorbing walls.

E 2.5.1.3. I/O Port parameters in the case of lumped sources/probes or lumped impedances

Lumped sources may serve two tasks. When they are placed in free space they represent radiating dipoles (see for example [UG 2.11.2 Step 5](#)). When they are placed between two pieces of metal they can serve as a lumped excitation of a structure (see the example of wire antenna presented in [UG 2.3.4](#)). Example of the *I/O Ports Parameters* dialogue in the case of a lumped source is presented in [Fig.E 2.5.1.3-1](#). We can see that the lumped source has a user defined internal impedance as well as the exciting field component. Note also that the considered source has been marked as NR:1. This means that it has been declared as one of the ports considered in S-matrix analysis (see [Section E 2.4.1](#) for reference). When the source is not supposed to take part in S-matrix analysis it would be marked as port NR:0.

The lumped probes act as the lumped sources with excitation equal zero. Thus it can serve as the elements probing field in free space or the resistive probes between two metal parts. They can be numbered as ports of S-matrix calculation or just serve as potential points for FD-probing (see the chapter on *Postprocessing*).

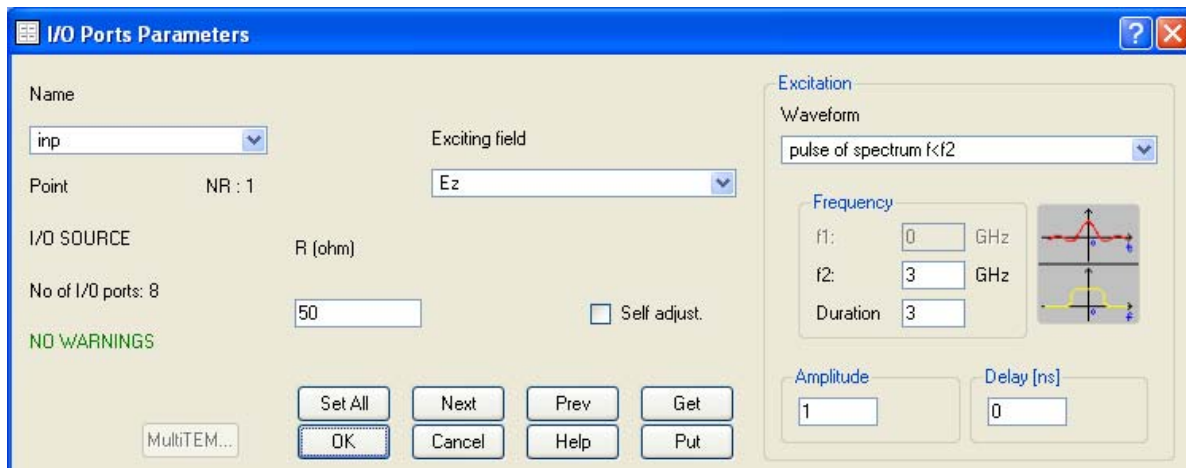


Fig.E 2.5.1.3-1 The relevant parts of the *I/O Ports Parameters* dialogue window in the case of a lumped source.

The *I/O Ports Parameters* dialogue for the case of a lumped impedance is presented in [Fig.E 2.5.1.3-2](#). Its application is described in the [Section UG 2.17](#). Here let us only point out that they can be connected to the considered structure in a similar way as the lumped probes. However they cannot be used as ports neither for S-parameter extraction nor for FD-probing.

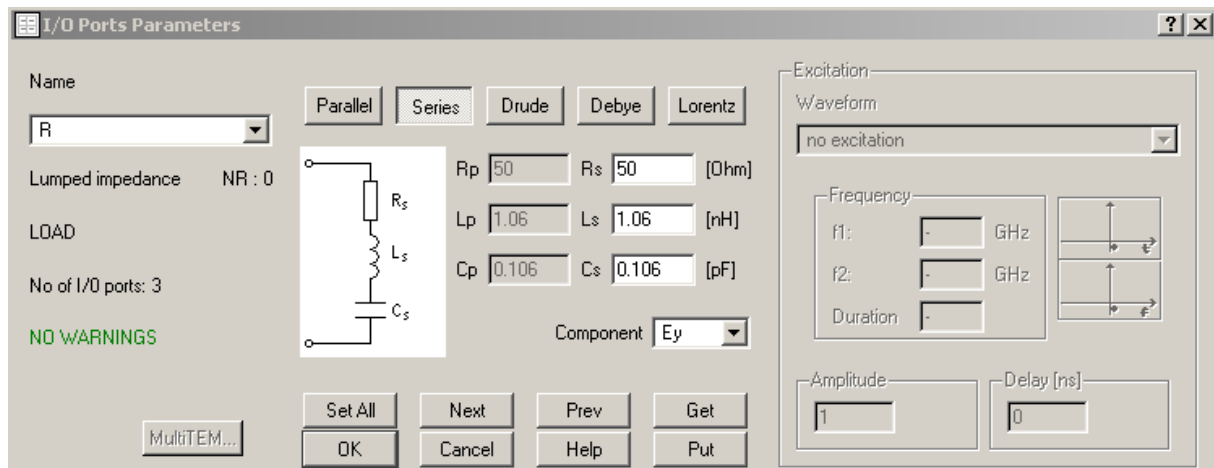


Fig.E 2.5.1.3-2 The relevant parts of the *I/O Ports Parameters* dialogue window in the case of a lumped impedance.

E 2.5.1.4. *I/O Port parameters in the case of free space excitation by a plane wave or a Gaussian beam*

Free space excitation by a plane wave or a Gaussian beam is a subject of several examples described in [Section UG 2.6](#). Here we will not repeat the technical discussions concerning such an excitation. Let us only point out that the plane wave is a limit case of the Gaussian beam when the neck diameter tends to infinity. The Gaussian beam has been introduced to model finite spot size illumination in optical or quasi-optical problems. The *I/O Ports Parameters* dialogue for the case of free space excitation is shown in [Fig.E 2.5.1.4-1](#).

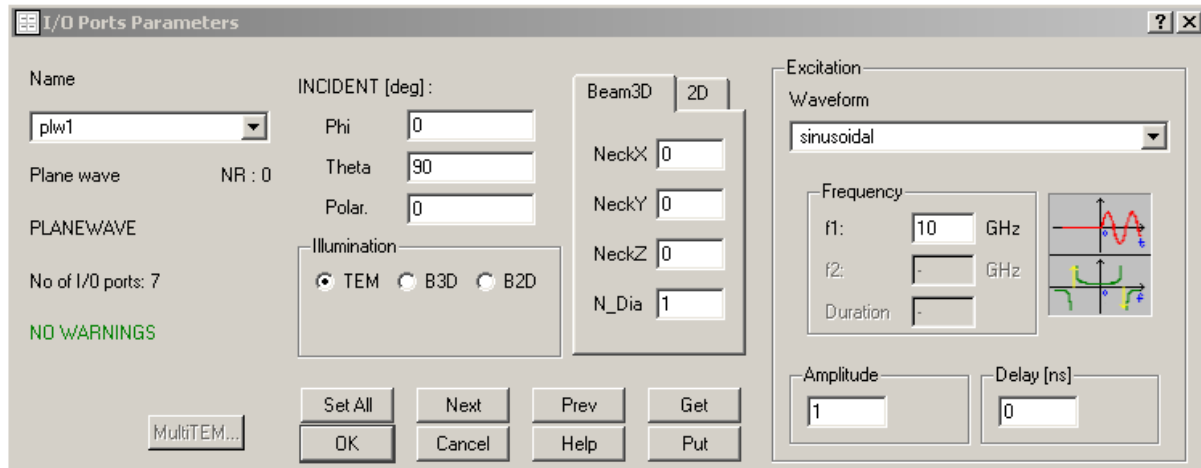


Fig.E 2.5.1.4-1 The relevant parts of the *I/O Ports Parameters* dialogue window in the case of a free space excitation by plane wave or Gaussian beam.

E 2.5.1.5. I/O Port parameters in the case of QW-V2D ports

In [E 2.1.9](#) we have discussed the basic *QW-Editor* rules in presentation of *QW-V2D* projects. The basic drawing takes place in 2D windows in cylindrical coordinates x, ρ . The basic assumptions of the *QW-V2D* simulation result in the following rules of application of ports:

- Transmission lines ports are applicable in a similar way as in the case of *QW-3D* projects. However the mode shapes are restricted to just one row of cells and the template generation problem becomes 1-dimensional.
- A vast majority of practical problems concern the excitation by a dominant waveguide mode or a TEM mode and thus the *Exciting field* in the *Ports parameters* dialogue should be chosen to be that of the mode *C-Ten1* or *TEM* respectively and the template is automatically generated.
- The generation of modes in arbitrarily filled axisymmetrical structure and with arbitrary angular variation is possible. For the users interested in such an option we recommend to try the template generation procedure based on description of *QW-3D* examples or ask the *QWED* support for more explanations.
- The *Mur* ABC works in a similar way as in the case of 3D projects. The *PML* ABC has not been implemented in *QW-V2D*.
- The interpretation of lumped sources or probes is somewhat different in *QW-V2D*. A point becomes a ring and thus a point source becomes a ring source with the angular distribution of current as declared in *Circuit Parameters*.
- The plane wave excitation options and the lumped impedances cannot be used in *QW-V2D*.

E 2.5.2. Processing/Postprocessing

Time-domain data, such as time evolution of fields or currents and voltages at lumped ports, can be accessed in *QW-Simulator* at any time and any location within the structure. Data in the frequency domain, such as S-parameters, is produced by means of the Fourier transform, which must be performed on selected time-domain quantities throughout the simulation. Additional memory must also be allocated at the beginning of the simulation process. For these reasons, the user must indicate the required frequency domain results before starting *QW-Simulator*. This is done in the *Parameters-Processing/Postprocessing* dialogue box as in [Fig.E 2.5.2-1](#).

Let us discuss the parameters describing frequency domain data, which will be extracted from the FDTD analysis. There are six types of *Processing/Postprocessing* algorithms and several additional options. Any or all of them can be selected for one simulation, and each has its own frequency band:

- *FD-probing*. At all I/O ports of the *lumped source/probe* type which have been marked as taking part in *FD-probing* (cf. [Fig.E 2.4.1-1](#)), the software will calculate Fourier transforms of terminal voltage and current, and therefore extract embedding impedance and admittance. If *contours* are defined in the *project*, field integrals along these contours will also be Fourier-transformed. The user specifies the frequency range of calculations. There are four main applications of this type of *postprocessing*:
 - Analysis of eigenvalue problems (resonant frequencies and field distribution of consecutive modes in arbitrarily-shaped and filled resonators). In this case a *Lumped source/probe* must be placed inside the resonator. It will serve as auxiliary excitation injecting energy into the structure through finite output impedance. We can watch the results of *FD-probing* calculations performed on the current flowing between the source and the resonator. With pulse excitation, minima of this current indicate resonant frequencies. The background theory of this approach has been explained in publications [\[14\]](#) [\[26\]](#). An example of application can be found in e.g. [Section UG 2.5.1](#).

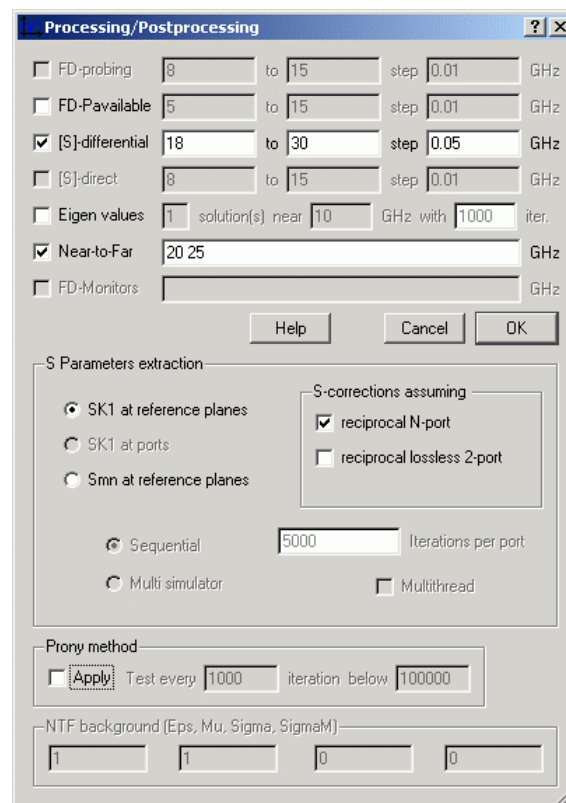


Fig.E 2.5.2-1 *Processing/Postprocessing* dialogue box.

- Analysis of embedding impedance for lumped elements. Typically such an analysis would be performed for lumped elements placed between two metal elements (wires, plates or blocks). These metal elements should be separated by one FDTD cell and a *Lumped source/probe* should replace the lumped element. Starting with version 5.0 a simpler option is also available: the source can be placed directly on the wire (without manually making a one-cell gap), and the software will cut the gap automatically, provided that the *Lumped source/probe* impedance is different than +INF. The *Lumped source/probe* of +INF impedance placed on the wire will not add any excitation or loading to the circuit, only mark the point where current flowing in the wire will be calculated by *H-field* loop integration. In the *I/O Ports Parameters*

dialogue, the *Exciting field* for this source should be selected as the *E*-field component along the replaced element (perpendicular to the metal surfaces). The *FD-probing* calculations will be performed on the current flowing between the source and the circuit, and on the port voltage (selected *E*-field integrated along the FDTD cell containing the *Lumped source/probe*). In the *QW-Simulator* we will see: the Fourier transform of the current and voltage, the embedding impedance and admittance versus frequency, and the port reflection coefficient (calculated with respect to the internal resistance of the *Lumped source/probe* if it is different from 0, and with respect to 50 ohm otherwise). Note that the calculated embedding impedance includes in parallel the capacitance of the FDTD cell into which the *lumped source/probe* has been inserted. This happens because the current considered by *FD-probing* comprises two components: current flowing into the metal embedding the circuit, and current flowing into the cell capacitance. If the latter component should not be considered in a particular application, the *[S]-differential* postprocessing should be selected instead of *FD-probing*. An example of application can be found in e.g. [Section UG 2.3.4](#).

- o Extraction of currents and voltages by integrating *H*-fields along a virtual loop surrounding a conductor and *E*-fields along a virtual line connecting two conductors. This information may be helpful in the analysis of quasi-TEM circuits for verifying the frequency range of their TEM behaviour. Also, lumped element models of parts of 3D structures can be extracted. Refer to [Section UG 2.13.2](#) for examples of application.
- o Analysis of currents induced in wires. In this case a *Lumped source/probe* should be placed on the wire. In the *I/O Ports Parameters* dialogue, the *Exciting field* for this source should be selected as the *E*-field component along the wire, and the *Lumped source/probe* impedance should be set to $R[\text{ohm}] = +\text{INF}$. *Excitation Waveform* will be ignored. The *FD-probing* calculations will be performed on the current flowing in the wire (*H*-field integrated around the *Lumped source/probe*).
- *FD-Pavailable*. This is a new functionality of version 6.0. It invokes calculations of the Fourier transform of *excitation waveform* of all sources. The result will be a square root of time-maximum spectral power density at each frequency point; when displayed in square scale, it will be time-maximum spectral power density in Watt. *FD-Pavailable* essentially amounts to applying *FD-probing* on source signals, and therefore both groups of results are accessed in *QW-Simulator* via the same *View-FD Probing Results* command. Refer to [Section UG 2.13.3](#) for examples of application.
- *[S]-differential*. We calculate the *S*-matrix of an *N*-port where *N* is the number of virtual ports defined in the structure. The system can include both *transmission line* ports and *lumped sources/probes*. For each transmission line port, the total fields at its respective *reference plane* are filtered through the modal *template* and decomposed into incident and reflected waves via the differential method. Reference impedances for *S*-parameter extraction are the actual frequency-dependent wave impedances of respective transmission lines. These impedances can be viewed in *QW-Simulator* by *Setup-Switch-Extended Results* option of *View-S Results* command. At a *lumped source/probe* the *[S]-differential* system operates similarly as the *FD-probing*. Thus for meaningful results, each *lumped source/probe* should be placed between two metal elements separated by one FDTD cell. The *[S]-differential* system considers the port voltage (selected *E*-field integrated along the FDTD cell), and the current flowing into the embedding circuit (without the current component flowing into the FDTD cell). Reference impedance is taken as resistance of the *lumped source/probe*, unless this resistance is zero, in which case 50 ohm is taken. Options of the *[S]-differential* postprocessing are described further in this Section. For detailed introduction to this postprocessing, refer to [Section UG 2.2.1](#), and to other subsections of [Section UG 2.2](#) for its examples of application to various types of transmission lines.

- *[S]-direct*. We also calculate the S-matrix, but without applying the differential decomposition. In version 6.5 of QW-3D this postprocessing is unavailable.
- *Eigen values*. This option is activating the optional frequency-domain eigenvalue solver. Interested users are encouraged to refer to *QProny* manual available on the installation CD.
- *Near-to-Far* transformation. We calculate the radiation patterns of a radiating structure, or scattering patterns of a scattering structure irradiated by a plane wave, at frequencies specified in this dialogue. This type of *postprocessing* is available only if *NTF box* has been previously defined in *QW-Editor*. Remember that for correct radiation pattern extraction, the *NTF box* must fully surround the structure and be itself surrounded by a homogeneous medium terminated with absorbing boundaries (except for symmetry and ground planes). The *NTF frequencies* can be introduced in any order and theoretically any number of them can be considered. However, note that collecting the NTF data for each new frequency (Fourier transforms of four field components in each cell on the NTF box) is quite consuming in terms of computer memory and computing time. A majority of *Near-to-Far* applications are in air, which is a natural environment for a majority of antennas. However, radiation patterns and antenna efficiency in dielectrics, and even in lossy dielectrics, may be of interest in e.g. biomedical studies. *Near-to-Far* postprocessing in QW-3D has been especially adapted to such cases in versions 5.0 and 6.0, respectively. The medium outside *NTF box* may be any *isotropic dielectric*. However, *Near-to-Far* will not automatically recognise the medium parameters. They must be explicitly repeated in the *NTF background* part at the bottom of the *Processing/Postprocessing* dialogue. Refer to [Sections UG 2.3.1 - UG 2.3.5](#) for examples of *Near-to-Far* applications in air, and to [Section UG 2.3.6](#) in other media.
- *FD-Monitors*. It performs Fourier-transformation of the six field components in selected sections of the circuit called *monitors*, at frequencies specified in this dialogue. In *QW-Simulator*, magnitudes of these field components as well as real and imaginary parts of the Poynting vector are accessible via *View-Fields Monitor* command. This way of watching the fields is complementary to their unconstrained *View-Fields* display. With *FD-Monitors* we need to pre-define sections and frequencies of interest, but then we obtain fields at multiple frequencies from one simulation with a pulse source. *Monitors* are set by UDO scripts from *elib/monitors* directory. *FD-Monitors* postprocessing is available only if at least one *monitor* has been set. Theoretically, any number of *monitors* and frequencies them can be considered. However, collecting the data for many frequencies and *monitors* may become quite consuming in terms of computer resources.

S Parameters extraction options. When choosing the option of S-parameter calculation we must decide upon the range of frequencies we are interested in, and the frequency step. The number of frequency points whereat the S-parameters are calculated increases to some extent the computing time and memory required. However, the computing time and memory used for S-parameter calculations at several hundreds of the frequency points usually does not exceed a few percent of the total computing time and memory needed to run the electromagnetic simulation. Thus the choice of this number in the range between one hundred and five hundred seems reasonable for most applications.

Note that the range of frequency in which we calculate the S-parameters cannot exceed the spectrum of the exciting pulse defined in the *I/O Ports Parameters* dialogue window. If this is the case, the S-parameters may be calculated with significantly increased numerical errors.

In the *Advanced* part of the *Processing/Postprocessing* dialogue box we have the choice of several options:

- *Sk1 at reference planes*. In this case we excite the circuit only from port number 1 and calculate the incident (a_k) and reflected (b_k) waves at each port. This permits to calculate $S_{k1} = b_k/a_1$. Note that:
 - The simple extraction produces results contaminated by numerical reflections from imperfect absorbing boundaries.

- If the circuit is declared as *reciprocal N-port*, it is possible to correct the value of S_{11} , applying the expanded formula $S_{11}=b_1/a_1-S_{12}a_2/a_1-S_{13}a_3/a_1-....$. The software enforces $S_{kl}=S_{lk}$ and calculates $S_{11}=b_1/a_1-b_2a_2/a_1^2-b_3a_3/a_1^2-....$
- If the circuit is declared as *reciprocal lossless 2-port*, it is also possible to correct the value of S_{21} , applying the formula $S_{21}=b_2/a_1-S_{22}a_2/a_1$. The software enforces $|S_{22}|=|S_{11}|$ and $\arg(S_{11})+\arg(S_{22})-2\arg(S_{12})=\pm\pi$.
- The software will not verify your declaration regarding reciprocity and losses, but it will ignore the declaration of *lossless 2-port* if the number of ports is different.
- *Smn at reference planes*. This is a general case in which we perform N simulations exciting the structure from N different ports, and then compose the complete S-matrix from N sets of partial results. Each of these sets corresponds to excitation from port i when the software calculates the parameters S_{ki} in the same way as the parameters S_{kl} are calculated in the *Sk1 at reference planes* postprocessing. When all N sets are available, the software corrects **all** the S-matrix elements for imperfect absorbing at **all** ports (reciprocity is irrelevant at this stage). Thus the final results of S_{mn} calculation may somewhat differ from the N sets of partial results. There are three regimes of running the *Smn at reference planes* postprocessing:
 - In the *sequential* regime the software performs N consecutive simulations (with excitation from each of the N ports). Each simulation lasts for a predefined number of FDTD iterations, specified by the user in the *Iterations per port* box. Note that during one simulation we obtain S_{kl} elements like in the *Sk1 at reference planes* postprocessing, and these parameters (either uncorrected or partially corrected for imperfect absorbing boundaries) are being displayed. Only after completing the N simulations with excitation from N consecutive ports QW-3D assembles the complete S-matrix, correcting all the S-matrix elements for imperfect absorbing at all port. Thus the final results of S_{mn} calculation may be somewhat different from the intermediate results displayed during calculation.
 - In the *multisimulator* regime, the software opens N instances of the *QW-Simulator*, with a different exciting port in each of them. An apparent disadvantage of this regime resides in the increased memory requirements. However, its important advantage is the possibility of the on-line monitoring of the full corrected S-matrix, calculated for the current number of FDTD iterations. This regime works on single-processor computers; its operation on multiprocessor computers is allowed only if the *QW-MultiSim* option has been acquired by the user.
 - The multithread regime is available only to the users, who have acquired the *QW-MultiSim* option. In such a case the software runs different instances on different processors, which decisively saves the computing time. Like in the multisimulator regime, increased memory is required. Users interested in this option are referred to the manual of *QW-MultiSim* available on the installation CD.

Attention:

Only in the circuit, which is not declared to be reciprocal, the reflection coefficient is calculated independently of the outputs. In other cases it is corrected for the reflection on the output ports. In those cases any errors of analysis at the output (wrong port definition, wrong template and so on) may strongly influence also the S_{11} calculation.

Prony method settings refer to the optional *QProny* software module. Basically, it reduces the computing time for circuits with sharp features in the frequency domain, such as narrow band pass filters. This is achieved by supplementing the Fourier transform calculations (naturally following time-domain evolution of signals which are physically slow in high-Q cases) with a digital signal processing by *Generalised Pencil of Function Method*. It allows one to extract the essential features of time signature based on the analysis of time samples obtained at the initial simulation phase. For details, consult a separate *QProny* manual.

NTF background shows current settings of the surrounding medium parameters (permittivity, permeability, conductivity and magnetic loss), which will be used by the *Near-to-Far* calculations. They can be modified with *Draw-Get-LibItem* command, applying *elib/actions/ntfbkg*. Parameters of the *NTF background* must be the same as respective entries seen for the surrounding medium in *Parameters-Media* dialogue. *QW-3D* does not check the consistency between the settings made here, and those in *Parameters-Media*. Inconsistent settings will lead to physically inconsistent postprocessing results. For examples of application refer to [Section UG 2.3.6](#).

E 2.5.3. Media parameters

The command *Parameters-Media* opens the dialogue window presented in [Fig.E 2.5.3-1](#).

Each of the *projects* has its own list of media. The list of the current *project media* is displayed on the right hand side of the dialogue window. On the left hand side we have one of *media libraries* in which we store the parameters of media for all our *projects*. These libraries are saved on the disk. We can pick one of them (the default one is called *default.mlb*), which becomes a so called *active library*. Note that a standard installation of *QW-3D* contains three media libraries: *standard.mlb*, *standard_with_food.mlb*, and *default.mlb*. At the beginning *default.mlb* is a copy of *standard_with_food.mlb*; if you are not going to analyse foods, please copy *standard.mlb* under the name *default.mlb*.

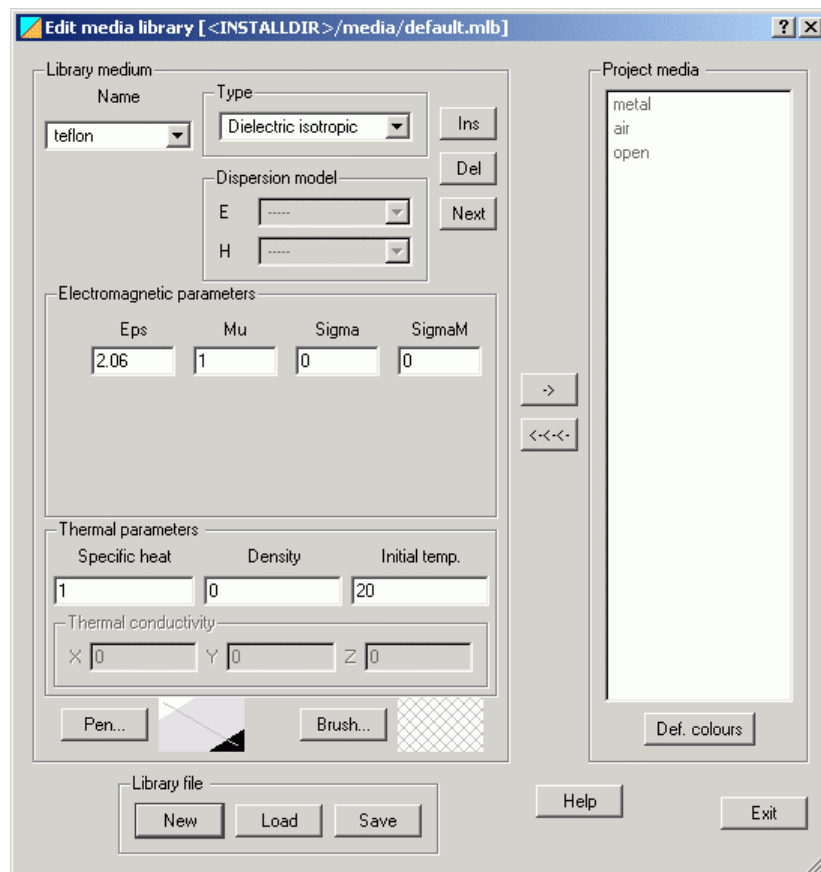


Fig.E 2.5.3-1 *Parameters-Media* dialogue window.

The name of the *active library* appears in the header of the window. One medium from this library is selected. Its name and parameters are displayed in the central and left part of the dialogue window. To see the whole list of the *active library* media, press the arrow box to the right of the name of the current medium. From this list we can select any other medium to display its parameters. These parameters can be modified.

Each non-dispersive medium can have up to fifteen parameters. Twelve of them describe its electromagnetic properties: permittivity, permeability, conductivity and magnetic losses along the three main axes of the coordinate system. Dispersive media have additional parameters describing the dispersion model. There are also three *thermal parameters*. *Specific heat* and *initial temperature* are relevant only if the QW-3D package operates with the optional QW-BHM module. *Density* is used in QW-BHM, and also for calculating Specific Absorption Rate (SAR).

The following media types are supported by QW-3D:

- *PEC* is an abbreviation for perfect electric conductor. In this case no parameters of the medium are needed.
- *Metalic* is a lossy metal. Magnetic permeability $Mu = \mu_r$ [dimensionless] and electric conductivity $Sigma = \sigma$ [S/m] can be declared. Note that although the meaning of these two parameters is exactly the same as in the case of media with volumetric losses, declaring a particular medium as a *metal with surface losses* enforces an absolutely different approach to its modelling in the FDTD analysis. Namely, *QW-Simulator* will not calculate the fields inside this metal volume. Instead, it will apply special surface impedance boundary conditions along the metal surface to represent the frequency-dependent skin-effect. Therefore this option should be used in the case of metals of finite, but very high conductivity. Details about the model and its implementation are discussed in references [32] [105] in [Chapter UG 3](#).
- *Dielectric isotropic*. Electric permittivity $Eps = \epsilon_r$ [dimensionless], magnetic permeability $Mu = \mu_r$ [dimensionless], electric conductivity $Sigma = \sigma$ [S/m] and magnetic loss $SigmaM = \sigma_m / (120 * \pi)^2 = \omega \mu_0 \mu_r \epsilon_r / (120 * \pi)^2$ [Ω/m] can be declared.
- *Dielectric anisotropic*. It uses the same parameters as *dielectric isotropic*, but different values can be set in different directions.
- *Ferrite* denotes a ferrite medium magnetised with a static *z*-oriented magnetic field. It has all the parameters of *dielectric isotropic*, and additionally the parameters of the ferrite model: damping constant *alfa* [unitless], saturation magnetisation M_s [T] and static biasing *z*-directed magnetic field H_i [A/m]. Details about the model and its implementation are discussed in references [90] [104] in [Chapter UG 3](#).
- *Dielectric dispersive* has magnetic permeability $Mu = \mu_r$ [dimensionless], electric conductivity σ [S/m] and magnetic loss $SigmaM$ [Ω/m] defined analogously as for *dielectric isotropic*. Its complex relative permittivity (including series losses) is given by *Debye*, *Drude* or *Lorentz* dispersion model with user-specified parameters:

$$\text{Drude:} \quad \epsilon_r(\omega) = eps\_inf + (2\pi f\_p)^2 / (j \omega 2\pi v\_c - \omega^2)$$

$$\text{Debye:} \quad \epsilon_r(\omega) = eps\_inf + (eps\_s - eps\_inf) / (1 + j \omega \tau)$$

$$\text{Lorentz:} \quad \epsilon_r(\omega) = eps\_inf + (eps\_s - eps\_inf) (2\pi f\_p)^2 / ((2\pi f\_p)^2 + j \omega 2\pi v\_c - \omega^2)$$

where:

eps_inf relative permittivity at infinite frequency [unitless],

eps_s static relative permittivity [unitless],

tau relaxation time [ns],

v_c inverse of the relaxation time (collision frequency) [GHz],

f_p pole frequency (plasma frequency) [GHz].

For *Debye* and *Lorentz*, $\epsilon_{s} < \epsilon_{inf}$ is not allowed, and $\epsilon_{s} = \epsilon_{inf}$ causes that the medium is considered as *dielectric isotropic* with frequency-independent permittivity ϵ_{inf} and conductivity Σ .

For *Drude* and *Lorentz*, $f_p = 0$ causes that the medium is considered as *dielectric isotropic* with frequency-independent permittivity ϵ_{inf} and conductivity Σ .

For *Debye*, $\tau = 0$ causes that the medium is considered as *dielectric isotropic* with frequency-independent permittivity ϵ_{s} and conductivity Σ .

There are also limits on the admissible level of losses with respect to the FDTD time step Δt . For *Debye*, the requirement is $\tau > 0.5 \Delta t$; for *Drude* and *Lorentz*, $(2\pi v_c)^{-1} > 0.5 \Delta t$.

- *Metamaterial* has electric conductivity $\Sigma = \sigma$ [S/m] and magnetic loss Σ_M [Ω /m] defined analogously as for *dielectric isotropic*. Its complex permittivity (including series loss) and complex permeability (including series magnetic loss) are given by *Debye*, *Drude* or *Lorentz* dispersion model with user-specified parameters. For permittivity, the same relations as above are used; for permeability, they become:

$$\text{Drude:} \quad \mu_r(\omega) = \mu_{inf} + (2\pi f_p)^2 / (j \omega 2\pi v_c - \omega^2)$$

$$\text{Debye:} \quad \mu_r(\omega) = \mu_{inf} + (\mu_s - \mu_{inf}) / (1 + j \omega \tau)$$

$$\text{Lorentz:} \quad \mu_r(\omega) = \mu_{inf} + (\mu_s - \mu_{inf}) (2\pi f_p)^2 / ((2\pi f_p)^2 + j \omega 2\pi v_c - \omega^2)$$

where:

μ_{inf} relative permeability at infinite frequency [unitless],

μ_s static relative permeability [unitless].

Limits on media parameters analogous to those for *dielectric dispersive* as above apply.

[Fig.E 2.5.3-1](#) shows the form of the window for *Dielectric isotropic* medium, in which case only three electromagnetic parameters and three thermal parameters are set. Variants of the window for *Metalic*, *Dielectric dispersive* with *Lorentz* dispersion model, *Metamaterial* with *Drude* electric dispersion and *Debye* magnetic dispersion, and *Ferrite* medium. The display for *PEC* is not shown since no parameters need to be specified.

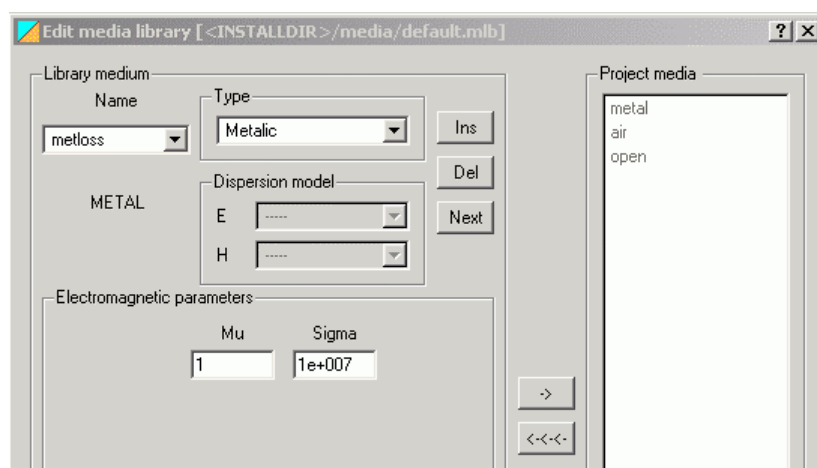
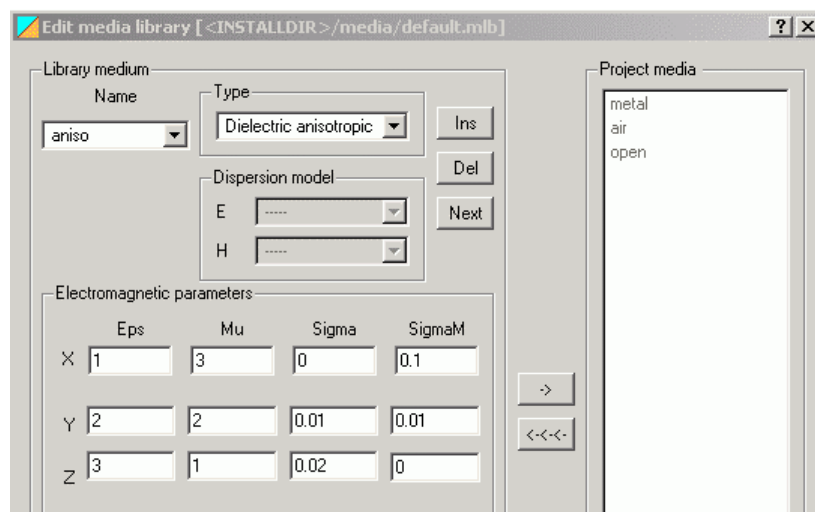
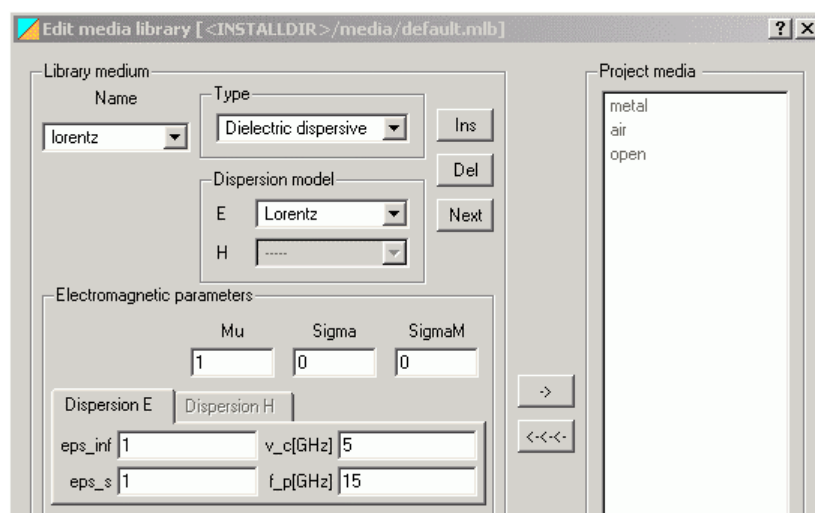
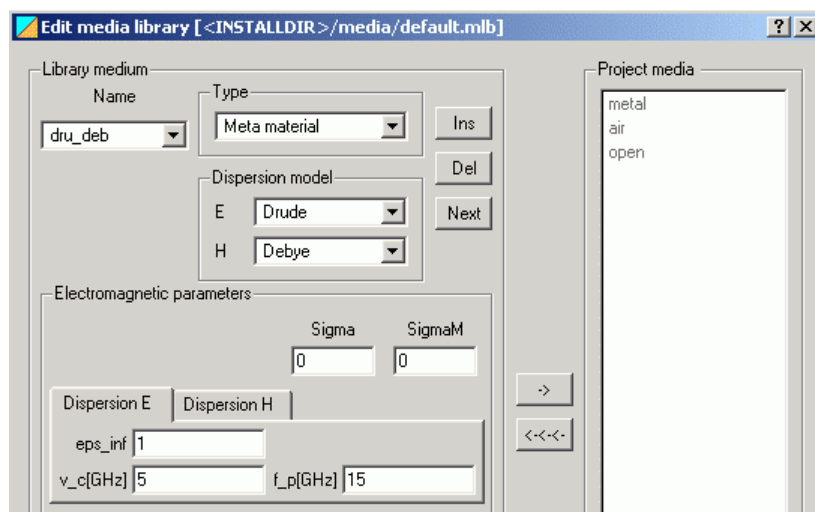


Fig.E 2.5.3-2 Parameters-Media dialogue window for *Metalic* medium.

Fig.E 2.5.3-3 Parameters-Media dialogue window for *Dielectric anisotropic* medium.Fig.E 2.5.3-4 Parameters-Media dialogue window for *Dielectric dispersive* medium.Fig.E 2.5.3-5 Parameters-Media dialogue window for a *Metamaterial*.

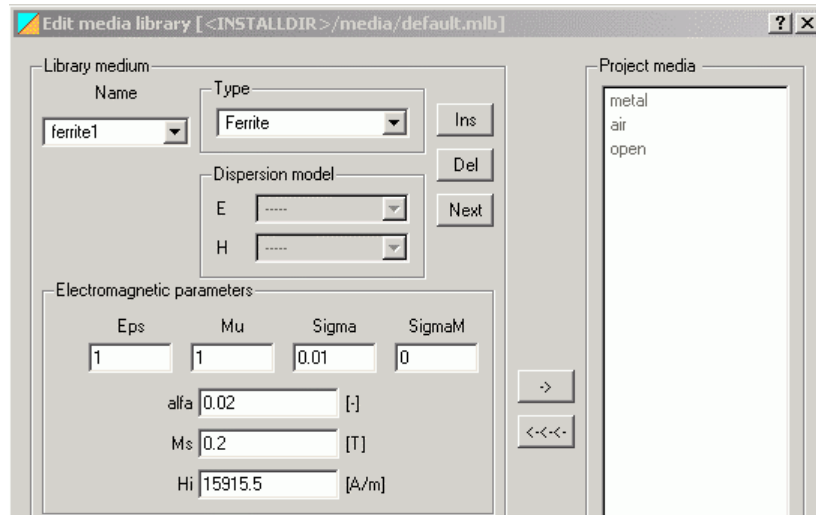


Fig.E 2.5.3-6 Parameters-Media dialogue window for a Ferrite medium.

Note the following meaning of abbreviations used in the dialogue:

- $Eps = \epsilon_r$ [dimensionless]
- $Mu = \mu_r$ [dimensionless]
- $Sigma = \sigma$ [S/m]
- $SigmaM = \sigma_m / (120 \cdot \pi)^2 = \omega \mu_0 \mu_r'' / (120 \cdot \pi)^2$ [Ω/m]
- $Density = \rho$ [g/cm³]
- $Specific\ heat$ [J/(g°C)]
- $Initial\ temp.$ [°C]

When solving Maxwell equations in lossy media, we multiply certain terms by a factor f_1 :

$$f_1 = 1 - 0.5 \cdot \sigma \cdot \Delta t / \epsilon \quad (E\ 2.5.3-1)$$

and we divide certain terms by a factor f_2 :

$$f_2 = 1 + 0.5 \cdot \sigma \cdot \Delta t / \epsilon \quad (E\ 2.5.3-2)$$

The terms f_1 and f_2 are responsible for attenuation of the fields. Thus for losses to be effectively considered by the FDTD implementation with floating point arithmetics, the following condition must be obeyed:

$$f_3 = 0.5 \cdot \sigma \cdot \Delta t / \epsilon > 1 \cdot 10^{-7} \quad (E\ 2.5.3-3)$$

Materials where $f_3 < 1 \cdot 10^{-7}$ are treated as lossless ones.

In the above relation, σ is conductivity in [S/m] and ϵ is real permittivity in [F/m] and Δt is FDTD time step (which can be checked in *Simulation Info* window).

The users interested in modelling very small losses have the following options:

Option 1 – recommended

Scale the problem by increasing σ so that relation (3) is obeyed. Note that small losses practically do not change the EM field patterns and dissipated power patterns scale linearly with medium conductivity. Thus if with $\sigma_2=100\sigma_1$ one obtains dissipated power density P_2 , then with σ_1 dissipated power density at the same point may be safely assumed as $P_1 = 0.01 P_2$.

Option 2

Try to apply coarser discretisation in space, causing larger value of FDTD time step Δt .

Parameters of the media can be modified. To the right from the media parameters table we have three additional commands concerning modification of the medium:

- *Insert* to clear the current medium display waiting for a new one to be introduced
- *Del* to delete the current medium
- *Next* to pick up next medium from the *active library* list

Another group of commands concerns operation on libraries. In the lower part of the dialogue window we have three commands:

- *New* for creating a new library
- *Load* for loading another library from a disk as an *active library*
- *Save* for saving the active library

Finally, we can see the *Pen* and *Brush* commands, which are used to assign colours to media. They affect the *QW-Editor* and *QW-Simulator* displays as follows:

- When an *element* made of a certain *medium* is drawn manually, *Pen* assigned to the medium determines the colour for displaying this element. However, subsequent changes of the medium assigned to this element are not automatically reflected in the changes of colour.
- When *external object* is drawn in *QW-Editor* (via the *Draw-Get-LibItem* command, or equivalently the  icon), *Pen* assigned to the medium determines the colour for displaying *elements* composed of this medium and included in the *external object*.
- When *TestMesh* operation is performed in *QW-Simulator*, each homogeneous subregion is displayed as determined by *Brush* of the medium, and meshing over this subregion is drawn in *Pen*.

One of typical operations performed in the *Parameters-Media* dialogue window is adding a particular medium to the *project*. To do this we select a medium from the *active library* and press $\rightarrow$ button to move it to the *project list*.

Sometimes we wish to update some media involved in the *project* without modifying the *libraries*. To start such an operation we should press $\leftarrow\leftarrow\leftarrow$ button. We see that the whole list of *project media* has replaced the current *active library* on the left hand side of the dialogue window. At the same time the window header has changed indicating that now, instead of the *active library*, we are dealing with the *project media list*. We can modify/delete/add media on that list. After the operation is over we should move the updated list back to the right hand side of the window. We can see that the button $\rightarrow$ has changed to $\rightarrow\rightarrow\rightarrow$. This means that now by pressing this button we will move to the *project media list* not just one item but the entire updated list. The $\rightarrow\rightarrow\rightarrow$ operation includes saving of the updated *project media list*.

E 2.5.4. Units

All the dimensions in the *project* are defined in the *units* chosen by the user via the *Parameters-Units* dialogue. There are four choices:

- ♦ metres
- ♦ millimetres
- ♦ microns
- ♦ nanometres
- ♦ inches
- ♦ mils

Note that the selected *units* will be applied:

- ♦ for user interface purposes – in all dialogues of *QW-Editor*, and also *QW-Simulator's* dialogue for virtual shift of *reference ports* for S-parameter extraction ([Section S 4.2](#)),
- ♦ internally by *QW-Editor*.

QW-Simulator operates internally in millimetres. Thus, as explained in [Section S 2.7.1](#), *QW-Simulator* will display the values of electric and magnetic field in [V/mm] or [A/mm], respectively. Conversion from the user-selected *units* into millimetres is performed by *QW-Editor* at export of *shape* files (*.sh3 – cf. [Section S 4.3.2](#)).

In principle, the choice of *units* is arbitrary, but in certain specific applications a limited dynamic range of *QW-Editor's* operations should be taken into account:

- You can write very long numbers into the *QW-Editor's* dialogues. However, all resultant coordinates (describing all created or modified *elements*, *lines*, *points*, *ports* etc.) will be instantaneously snapped by *QW-Editor* to 5 digits after decimal point (i.e., 10^{-5} in current *units*).
- In *project* files (*.pro) saved by *QW-Editor* upon the *File-Save* command, all coordinates are written with the accuracy as above.
- In *shape* files (*.sh3) saved by *QW-Editor* upon the *File-Export* command, all coordinates are written with the accuracy of 15 significant digits in millimetres always (in exponential form if necessary).

The above limitations should not restrict flexibility of applying the software or its overall accuracy, as long as you proceed in a reasonable way, and do not try to define a circuit that is really one micron thick at the level of one thousand inches. As a safe guideline, we recommend such choice of the units that the *project* does not exceed below -20000 units or above +20000 units in any dimension. Note that the above snapping of coordinates has been introduced in purpose, to avoid hazards with declaration of shapes. In other words we are trying to limit the risk of introduction of unintentional slots or overlapping appearing in the considered structure due to limited accuracy of computer floating point operations on geometrical dimensions.

E 2.6. Application of *QW-Editor* using library objects

In *QW-3D* version 2.1 QWED has introduced a new system of *User Defined Libraries*. Although the *UDO* scripts have been used in the earlier versions, now they are supposed to be used in a different and much more user-friendly way. The new structure of libraries has been described in [Sections UG 2.8.1](#) and [UG 2.8.2](#). There are also other examples of the library applications throughout the entire [Chapter UG 2](#).

More advanced users who wish to build their own libraries are encouraged to study [Chapter E 3](#) in which the syntax of the UDO language has been presented.

E 2.7. File exporting and starting the simulation from *QW-Editor*

E 2.7.1. Export and Run commands

Let us take a look at the files in which our example *project* *project_name* has been stored. To load it to the *QW-Editor* we have read the file *project_name.pro* containing the information about the shape of the structure and the parameters. To prepare the use of the *QW-Simulator* the *QW-Editor* performs the following tasks:

- It defines the sequence of operations to be executed by the *QW-Simulator* to perform the simulation according to the set of parameters. As a result a *tasker file* *project_name.ta3* is created. This is the *project tasker file* or in other words a text file containing the commands for the *QW-Simulator*. A closer look at the directory in which the *wgtocx1.ta3* has been created reveals that there is also another *tasker file* *project_name0.ta3*. This is a copy of the *project_name.ta3* with one exception: instead of generating the mode *templates* the *QW-Simulator* is asked to read the previously generated *templates* from disk. This allows us to avoid generating the *templates* when we know that they are the same as the ones previously calculated and saved on the disk.
- It prepares the files containing the description of the analysed structure divided into FDTD cells. Those files are called: *project_name.sh3* for the entire 3D structure, and auxiliary files describing the consecutive ports, each of them called by the *port* name with the extension **.sh3*.
- It prepares the files containing the description of the parameters of FDTD simulation. Those files are called: *project_name.pa3* for the parameters of 3D analysis, and auxiliary files describing the parameters of the consecutive ports, each of them called by the *port* name with the extension **.pa3*. However, note that the *project_name.pa3* file will not be created if we select *Smn postprocessing*. Instead, for the *N*-port circuit *QW-Editor* will export *N* different **.pa3* files, instructing *QW-Simulator* to apply excitation to the *N* consecutive ports. Each of these files will be called by the name of the *reference plane* corresponding to the excited *port*.

All the above operations are performed upon execution of the *File-Export* command. After the *export* has been performed we can leave the *QW-Editor*, invoke the *QW-Simulator* and perform the simulation. This kind of operation may be advantageous on computers with small memory. It is also possible to run the simulations directly from the *QW-Editor* by invoking the command *File-Export&Run* (or the shortcut flash icon corresponding to it). However, during such a mode of operation more memory will be occupied by the program itself and less of it left for storing variables of the electromagnetic modelling.

In this section we have signalled that the *QW-Editor* produced several types of files, which are needed by the *QW-Simulator*. More systematic description of these files and other files used by the *QW-3D* package can be found in [Chapters E 4](#).

E 2.7.2. How to assure the proper mesh generation

The process of meshing consists of generating a set of FD-TD cells describing the considered structure. As it has been mentioned in [Section UG 2.1](#) half-cells form sublayers in XY plane (with one layer of the FD-TD cells composed of two sub-layers). Each of the half-cells of *QW-3D* conformal FD-TD method must be a cuboid composed of one medium or a cuboid divided by a plane into two parts filled by different media. Meshing is done automatically by *QW-Editor* just before the *Export*

operation. *QW-Simulator* reads the meshing information coded in *.sh3 file and assigns to each of the cells proper electrical parameters (depending on the cell's shape and media filling).

As it was mentioned the meshing is performed automatically by *QW-Editor*. However there are certain important rules to be obeyed while introducing the geometrical data. We need to provide to the mesher consistent information leading to the expected results of the meshing process. Here we would like to draw the user's attention to several situations, which may cause meshing problems and thus should be avoided in *projects*. They first concern intersecting elements, multiple-filled cells, and meshing hazards along the Z-axis.

Intersecting elements

Important definition:

Two elements are defined to be non-intersecting if they obey the following condition:

In each of the cross sections parallel to the XY plane the pair of the elements is either disjoint or one of the elements is entirely included in the other.

If an element A is included in element B but their borders partially coincide the elements can be treated as non-intersecting only if the element A is defined as "filled inside" by a particular medium while the element B as "filled outside" by this or other medium.

Intersecting elements must be treated by the *Edit-Join* operations. These operations produce out of them new elements that are non-intersecting. If the intersecting elements are still present during the mesh generation by *QW-Editor*, the mesh generator may not treat them in accordance with the user's intention. The resulting analysis will then be erroneous as exemplified in the example of [Section UG 2.12.1](#). In general it cannot be avoided that it is the user's responsibility to provide consistent data to the editor and thus to eliminate intersecting elements before performing Export.

Note that according to the above definition if an element is internally tangential to another one it should in general be treated as intersecting. However some of such situations (described in [Section UG 2.12.2](#)) are automatically detected by the *QW-Editor* and treated correctly without the Join operation.

Note: In case of any doubts if the geometry of the project was correctly defined by the user and correctly understood by the mesh generator we strongly recommend to verify the meshing results using Test-Mesh option of QW-Simulator extensively used in examples of [Section UG 2.12](#) and described in [Section S 2.6](#).

Multiple-filled cells

As it was mentioned above *QW-Editor* creates and exports to *QW-Simulator* a conformal mesh that may include cells filled with two different media (two dielectrics or metal and dielectric). In the case of complicated geometries, the mesh generator may initially create cells filled with three or more media. We shall call them *multiple-filled cells*. *Multiple-filled cells* will be simplified according to the following rules:

- If the *QW-Editor* finds a cell shared by a metal and two different dielectrics, it assumes that the cell is shared between the metal and the dielectric occupying larger part of the cell (replacing with it also the other dielectric).
- If the *QW-Editor* finds a cell shared by three different dielectrics, it assumes that the cell is shared between the two dielectrics occupying larger parts (and neglects the presence of the third dielectric).
- In the case of cells occupied by more than three media the *QW-Editor* will assume that the cell is shared between two media occupying two largest parts.

These simplification rules are still under intensive tests. Thus temporarily the user is encouraged to use *Test Mesh* option in *QW-Simulator* to verify the mesh created in regions of complicated geometry. In general, since the *QW-Simulator* treats more accurately cells filled with one or two media, it is recommended to avoid *multiple-filled cells* by defining *mesh snapping planes* passing through sensitive junction points.

Meshing hazards along the Z-axis

Let us recall that at the *level* of *bottom* and *cover* of each *element* the software automatically creates *mesh snapping planes* (cf. *Setup-Mesh* command, [Section E 2.2.1](#)). If the mesh generator detects two such *planes*, enforced by two different *elements*, at different *levels* but very close to each other – it will create a very thin layer of cells. What may happen in some specific cases is that these two *levels* should really be identical, and they have been shifted apart as a result of previous operations on *elements*. The difference may also arise when we have two *elements*, at the same physical *level* and of the same physical *height*, but one of them is *simple* while the other - *combined*. For the *combined element* the *levels* of *bottom* and *cover* are explicitly remembered by *QW-Editor*. For the *simple element*, *QW-Editor* remembers its *height* and the *level* of *bottom*, and calculates the *level* of *cover* therefore. The result may be different from the *level* of *cover* of the *combined element*.

The thin layer of cells created in such cases would be unnecessary, prolonging the simulation time and deteriorating the accuracy. To avoid such problems, *QW-Editor* precedes the actual mesh generation by snapping of all coordinates to 10^{-6} in current *units* (cf. [Section E 2.5.4](#)). This measure has proven sufficient in all typical applications, but nevertheless the user is advised to consult the *Mesh-Splanes info* ([Section E 2.2.5](#)) whenever substantial changes in the *project* have been made.

E 2.8. Command line options of QW-Editor

It is possible to call *QW-Editor* from a command line with parameters. This feature is prepared for use with optimisers of 3D structures. Starting with version 6.0, QWED offers its own *QW-OptimiserPlus*, integrated with *QW-Simulator*. The old external *QW-Optimiser* also remains operational. However, interested users may also try to couple *QW-3D* with their own optimising procedures. The following options are possible:

- **-pproject_name** Starts the *QW-Editor* loading the specified *project*.
- **-i** Enforces iconised operation of *QW-Editor*.
- **-e** Instructs the *QW-Editor* to *export* the *project*.
- **-q** Quits the *QW-Editor* (reasonably, this should be preceded by *export*).
- **-m** Instructs the *QW-Editor* to perform the *Select Object-Modify* operations on the *object* which has the same name as *project*. For reasonable application, the corresponding **.udo* file describing this *object* (or some other **.udo* and **.txt* files called from within this file) should be previously modified by the user / optimiser.
- **-onumber_of_iterations** Instructs the *QW-Editor* to further instruct the *QW-Simulator* to run the analysis for a specified number of iterations. The *QW-Editor* will comply by exporting modified files (modifications regarding *RunIter* instead of typical *Run* tasks, specified format of files for saving the results).

E 3. Syntax of the User Defined Object (UDO) language

E 3.1. What is the User Defined Object

There are three basic advantages of application of parameterised objects in electromagnetic design:

- In the vast majority of applications engineers design structures, which can be composed of typical primitive objects by simple putting them in proper place or by exercising Boolean operations on them.
- In many practical cases design engineers deal with one type of a microwave structure for a long time. Not changing the general shape, they modify its dimensions to optimise performance or to redesign the device for a different set of parameters or a different frequency band. In such a case the most convenient way to proceed is to define the investigated structure as *User Defined Object* or as a set of such *objects* with some dimensions introduced as parameters. To change one of these dimensions it will be sufficient to call the *object* once more and to introduce new values of the parameters - the operation which can be accomplished in seconds.
- Parameterized *UDOs* open new horizons for running *QW-3D* in an automatic optimisation loop using *QW-OptimiserPlus*, the old *QW-Optimiser* or other external optimisers like for example those available in MATLAB TOOLKIT (see [Section S 3.7](#)).

The *QW-3D* package includes *QW-ObjectGenerator*. *QW-ObjectGenerator* is essentially an interpreter for source programs provided by *QWED* in a form of library *UDOs* or prepared by the user in a specially developed *UDO* language. These source programs are stored in files with **.udo* extension. It is recommended that each **.udo* file contain a description of one *User Defined Object*. The *UDO* language is simple and yet gives the user the possibility to create his own arbitrarily complicated parametric *objects*, best suited for his particular requirements.

It should be noted that *QW-ObjectGenerator* is incorporated into the *QW-Editor*. This means that while inserting a new *object* or modifying the existing one, the user introduces its parameters via a dialogue window, which automatically appears in *QW-Editor*. Also any *User Defined Object* interpreted by the *QW-ObjectGenerator* naturally appears in all open windows of the *QW-Editor*.

This Chapter describes the syntax of the *UDO language*, explains the structure of *UDO* programs, and contains several examples of *User Defined Objects*, which are also included on the installation CD of *QW-3D*.

E 3.2. Structure of a single UDO and calls to other UDO's

Let us consider a very simple *UDO* script *verstepa.udo*, which can be found in the directory *..eliblexamples*. It is a simplified version of *verstep.udo* considered in the example of [Section UG 2.8.2](#). In *QW-Editor* after clicking over , choosing the library *examples*, clicking with the right mouse button over the *object verstepa* and choosing *Edit UDO* we obtain a display of the *UDO* script as shown below.

```
comment="Vertical coaxial line with impedance step";
bitmap="verst0.bmp";

PAR("object name",oname,"vrs");
PAR("inner dia",ind,3.04);
PAR("outer dia1",oud1,7);
```

```

PAR("outer dia2",oud2,9);
PAR("length of I/O segment",l1,15);
PAR("length of middle segment",l2,30);
PAR("input-to-reference dist",rdist,5);
PAR("I/O port param. file",pf,NO);
PAR("number of sections",nsec,32);
ENDHEADER;

TEST((ind>0)&&(oud1>0)&&(oud2>0),"Diameters must be positive");
TEST((l1>0)&&(l2>0),"Lengths of sections must be positive");
TEST((oud1>ind)&&(oud2>ind),"Inner dia must be smaller than the outer ones");

OPENOBJECT(online);
# calls to three coaxial line segments
CALL("coax2/cxv1d.udo",seg1,l1,ind, oud1,air,nsec,x,y,z-l1,10);
CALL("coax2/cxv1d.udo",seg1,l2,ind, oud2,air,nsec,x,y,z,10);
CALL("coax2/cxv1d.udo",seg3,l1,ind, oud1,air,nsec,x,y,z+l2,10);
CLOSEOBJ;

```

The above script can be considered as an example of *UDO* programs. Each of them is composed of two parts:

- *header* - contains the information that will appear in the header display of the *UDO*, namely:
 - a comment line - describing the object to be displayed in both library header and the *UDO* header.
 - bitmap file name – indicating the name of the file containing a 110x110 pixels bitmap to be displayed in the *UDO* header. The software looks for the bitmap first in the directory of the considered *UDO* and then in the directory *.. elib\bitmaps*. If a proper bitmap is not prepared, *nobitmap.bmp* name is recommended. Otherwise at each *object* call the program will stop with a warning about a missing bitmap.
 - parameters – the list of parameters to be later modified from the header containing the parameters description (to appear in the *UDO* header), its internal name in the *UDO* script and the initial value.
- *body* – contains all other commands not included in the header. Between *header* and *body* **ENDHEADER** separator is necessary. Commands available for application in the *body* will be described in this and subsequent sections. Below we will discuss only those used in *verstepa.udo*:
 - **TEST** command – is not obligatory but helpful. It helps to avoid entering unreasonable or inconsistent parameters. When the condition in the *TEST* argument is not met the commands listed in the *UDO* script after the *TEST* are not executed and a warning is issued. Thus it is convenient to put the *TESTs* just after the header.
 - **CALL** command – executes a call to another *UDO* (to a *nested object*). In our case we have three calls to the same “*coax2/cxv1d.udo*” (describing one segment of coaxial line) but with different parameters. The list of parameters includes: the path to the called *cxv1d.udo* + all parameters of the *cxv1d.udo* header + the position of the called object (x,y,z) in the co-ordinates of our *project* + the total number of parameters). This way we generate three segments of coaxial line as shown in [Fig.UG 2.8.2-1](#). As requested in the calls the consecutive segments will be named seg1, seg2 and seg3.
 - a frame: **OPENOBJECT(name).....CLOSEOBJ** - All the objects or elements generated within this frame will belong to an object *name*. For example let us draw *verstepa.udo* with default parameters (including the *name=vrs*) in an empty *project* and press *Select-Object*. On

the displayed list we will see that there is one object of the type *udo* named *vrs*. When this object is highlighted we can press over *Info* to verify that it includes three *nested objects*: *seg1*, *seg2* and *seg3*. This is correct since those objects were generated by *CALL* commands within the frame of *vrs*. Double click over *vrs* name on the object list brings up the header of *verstepa.udo* where we can change its parameters. Note that highlighting for example object *seg1* and pressing over *Info* brings the information that *seg1* contains two other objects: *seg1diel* and *seg1center* (since *cxv1d.udo* used to generate *seg1* included two calls to other UDOS). However all the objects listed in our *project* except for *vrs* are marked as of the type *normal* which means that they are *nested objects* and we cannot change their parameters from here. They can be selected, marked moved and even deleted but their parameters are not accessible from this level.

- o # - marks a comment line. All commands from # to the end of line will be neglected by QW-Editor's Object Generator

Syntax of introduced UDOS commands

- **PAR** (<description>, <name>, <default value>);

where:

<description> - max. about 70 characters of parameter description. If the description contains operator chars, the text has to be delimited with apostrophes.

<name> - case sensitive name of parameter which can be used in expressions within the *body*.

<default value> - initial value for parameter, it can be a number or string.

Function *PAR* can be used in the *header* only.

- **TEST** (<logical expression>, <message text>);

<logical expression> - expression which should be true,

<message text> - text displayed if logical expression is false.

- **CALL** (<path>, <par1>, <par2>, ..., <parn>, <parx>, <parx>, <parz>, <N>);

<path> - path to *.udo file for *nested object* (the path is indicated with respect to the root directory of *elib*)

<par1>..*parn* - *n* parameters of *nested object*, in the place of each parameter we can put a value, a variable or an arithmetic expression,

<parx>, <parx>, <parz> - origin of *nested object*,

<N> - number of parameters of this *CALL* excluding <N> (*N*=*n*+4)

Additional remarks

- o ; - semicolon is a very important separator. All instructions (with a few exceptions specifically described later in this Chapter) must be ended with semicolon. Its omissions produce errors of interpretations of the UDO scripts. Thus verification if each of the instructions is ended with semicolon should be the basic procedure of the UDO script debugging.
- o One line of the UDO script can contain several instructions separated by semicolons.
- o The carriage return sign can be brake an instruction into two or more lines. However it cannot break the names of commands, parameters and variables).
- o UDO interpreter is case-sensitive (it differentiates between upper- and lower-case characters).

- o UDO not containing *OPENOBJECT(name).....CLOSEOBJ* frame is valid and can be drawn. However it cannot be modified since its name does not appear on the *Select Object* list.
- o UDO with more than one *OPENOBJECT(name).....CLOSEOBJ* frame will be correctly drawn for the first time. However it will not be possible to modify it correctly. Thus such a structure should be treated as incorrect.

E 3.3. Defining elements, combined elements and materials

Let us consider *..lexamples\threeel.udo* presented below:

```
comment="example of three elements";
bitmap="threeel0.bmp";

PAR("name", oname, "3el");
PAR("length ", a, 6 );
PAR("width ", b, 5 );
PAR("height of first elem.", h1, 5 );
PAR("height of second elem.", h2, 7 );
PAR("height of third elem.", h3, 3 );
PAR("shape ratio top/bottom", sr, 0.5);
PAR("medium", med, air );
ENDHEADER;

TEST( (a>0)&&(b>0)&&(h1>0)&&(h2>0)&&(h3>0), "Dimensions should be positive" );

OPENOBJECT(oname);
ELEMENT( z, h1, 0, med, cubic1,IN);
  NEWLINE(x-a/2, y-b/2, x+a/2, y-b/2);
  ADDLINE(x+a/2,y+b/2);
  ADDLINE(x-a/2,y+b/2);
  CLOSELINE;
ENDELEM;

ELEMENT( z+h1,0, 5, med, prismsb,INe);
  NEWLINE(x-a/2, y-b/2, x+a/2, y-b/2);
  ADDLINE(x+a/2,y+b/2);
  ADDLINE(x-a/2,y+b/2);
  CLOSELINE;
ENDELEM;

ELEMENT( z+h1+h2,0, 6, med, prismt,INe);
  NEWLINE(x-sr*a/2, y-sr*b/2, x+sr*a/2, y-sr*b/2);
  ADDLINE(x+sr*a/2,y+sr*b/2);
  ADDLINE(x-sr*a/2,y+sr*b/2);
  CLOSELINE;
ENDELEM;

ELEMENT( z+h1+h2, h3, 0, med, cubic3,INn);
  NEWLINE(x-sr*a/2, y-sr*b/2, x+sr*a/2, y-sr*b/2);
  ADDY(sr*b);
  ADDX(-sr*a);
  CLOSELINE;
ENDELEM;
CLOSEOBJ;
```

Please draw the above *UDO* in an empty *project* to see that it produces two cuboids connected by a prism. Thus we have three elements declared in the *UDO* script. Two of them are simple *elements* and the connecting prism is a *combined element* as defined in [Section UG 2.1](#). We will discuss here the basics of introduction of elements in *UDO* scripts.

- A simple *element* starts with a key word **ELEMENT(...)**; and ends with a key word **ENDELEM**; There are six parameters to be defined in the **ELEMENT(...)**; as specified in the syntax description at the end of this section.
- Note that first two parameters define the position of the bottom of the element and its height. The subsequent instructions are for drawing a contour of the element in *XY*-plane. Thus we have the commands **NEWLINE(x1,x2,y1,y2)**; **ADDLINE(x2,y2)**; **CLOSELINE**;. It is very important to close the line. Lines not closed may cause that during the meshing process the medium will “pour out” of the contour making the entire meshing incorrect. Note that if the drawn line is parallel to X or Y axis **ADDLINE(x2,y2)**; can be replaced by simplified commands with incremental argument: **ADDY(dy)** or **ADDX(dx)**. This is exemplified in the definition of the last element (*cubic3*) which is much simpler than the definition of the first element (*cubic1*).
- The consecutive points of the contour of the element **must be left (counter-clockwise) oriented**. Violation of this rule may result in confusion about what is placed inside the contour and what is placed outside it. Thus please be careful about that.
- The third parameter indicates the type of element. In the case of a simple *element* this parameter should be set to 0. In the case of a *combined element* we use a pair of frames: **ELEMENT(...)**; ... **ENDELEM**;. The first one describes the bottom of the combined element (and its type is 5), the second describes the top of the combined element and its type is 6. Thus in the *threeel.udo* we have three elements but four frames **ELEMENT(...)**; ... **ENDELEM**;. The first one declares the element *cubic1* (name of the element is the 5-th parameter), the following two - a combined element *prism* (with the *bottom prismb* and the *top prismt*) and the fourth one - an element *cubic3*. Note that:
 - Both bottom and top of combined elements have the height equal 0. The actual height of the element is defined by the difference between the levels of the bottom and the top.
 - The declaration of the top must immediately follow the declaration of the bottom.
- The 4-th parameter indicates the name of the medium of which the element is made. The name of medium must be one of those used in the *Medium Library* of the current *project*. The assumed parameters of the medium are those found in the *Medium Library* unless they are overwritten in the *UDO* script by an instruction **MEDIUMPAR(...)** that gives new values of the parameters for the particular medium.
- The 6-th parameter indicates the *orientation* of the contour of the element. **IN** indicates that declared medium (4-th parameter) is placed **inside** the defined left (counter-clockwise) oriented contour. **This is a typical choice and practically the only one recommended to our users**. Theoretically we have an option **OUT** placing the declared medium outside the left-oriented contour but this is an option introduced for some very special applications and not recommended for general use.
- It may be noticed that while the first element has the *orientation* **IN** the second one - *orientation* **INe** and the third one – *orientation* **INn**. Actually the declaration of **IN** and **INe** gives the same result – the top and bottom of the element enforces an *E-plane* of FD-TD grid as defined in [Section UG 2.1](#). With the declaration of **INn** the bottom and top of the element enforces a *neutral* mesh snapping plane which means that there will be *either E-plane or H-plane* of the FD-TD grid. The result of these declarations is visible on the list of elements presented below (invoked by *Select-Element*). Two first elements are marked with *E* and the third one with *N*.

	Name	Z level	Height	Type	Medium	Status
1	cubic1	0	5	Element E	air	
2	prismb	5	(7)	Combined(bottom) E	air	
3	prismt	12	(-)	Combined(cover) E	air	
4	cubic3	12	3	Element N	air	

Syntax of introduced above and similar UDOs commands

- **ELEMENT** (<level>,<height>,<type>,<medium name>,<name>,<spin/wire>) - Start of element frame, where:

<level> - level of *element* base, usually relative to predefined origin variable *z*,

<height> - height of *element*, for combined elements is forced to zero,

<type> - type of element:

0 – simple *element*,

5 - *bottom* of combined element,

6 - *cover* of combined element.

<medium name> - name of element medium (if there is no such medium air is forced),

<name> - max. 20 characters name of element,

<spin/wire> - orientation of element edge or wire marker:

IN, INe, INn – medium fills inside the defined left (counter-clockwise) oriented contour. These are typical settings practically for all applications except of wires. **IN** and **INe** enforce *E-plane* of the FD-TD grid at the level of the bottom and top of the element. **INn** enforces neutral plane (either *E-plane* or *H-plane*) of the FD-TD grid at the level of the bottom and top of the element. Additionally lowercase letter **d**, **l** or **h** can be appended to **IN** option to define classes of *AMIGO* priorities (respectively **d**isabled, hard edges (**l**ines), **h**ard).

OUT - medium fills outside the defined left (counter-clockwise) oriented contour.

Note that this option is not recommended for a general use.

WIRE - element is a wire line. The programmer has to ensure a proper shape of such an element. The line describing the wire shape does not need to be closed. Inside the frame of the *element* declared as **WIRE**, there may be a declaration of its diameter as presented below.

- **ENDELEM** - End of element frame.
- **SETATTR**(<which_attribute>,<attr_value_1>,<attr_value_2>) - Sets a property for element, allowed only inside the frame of the *element* element frame, where:

<which_attribute> - index of attribute to set:

index **0** - *AMIGO* status:

<attr_value_1>

-1 - no change, **0** - soft (default), **1** - hard edges, **2** - hard

<attr_value_2>

-1 - no change, **0** - enabled (default), **1** - disabled.

- **DIAMETER**(<dia>) - Declaration of *wire* diameter, allowed only inside the frame of the *element* declared as **WIRE**.
- **NEWLINE** (<x1>,<y1>,<x2>,<y2>) - Insert line from point (x1, y1) to point (x2, y2).
- **ADDLINE** (<x3>,<y3>) - Insert line from previous point to point (x3, y3).
- **ADDX**(<dx>) – Insert a line parallel to X-axis from (x2,y2) to (x2+dx, y2).
- **ADDY** (<dy>) – Insert a line parallel to Y-axis from (x2,y2) to (x2, y2+dy).
- **ADDXYZ**(<x>,<y>,<z>) - Insert a contour point, to be used only in the group of library “contour” objects.
- **CLOSELINE** - Insert line from previous point to (x1, y1) point of the last **NEWLINE**.
- **INSERTMEDIUM**(<mediumname>, <type>) – Inserts a new medium into the *project*, if medium of the same *name* does not exist. The parameter *type* options are: *ISOTROPIC*, *ANISOTROPIC*, *DIELECTRIC_DISPERSIVE*, *FERRITE*, *METAL*, *PEC*, *PMC*.
- **MEDIUMPAR**(<mediumname>,<epsx>,<mux>,<sigx>,<msigx>,<epsy>,<muy>,<sigy>,<msigy>,<epsz>,<muz>,<sigz>,<msigz>,<density>) - Change the parameters of the medium “mediumname” to those specified by the subsequent parameters. These parameters should be self-explaining by comparison with those displayed in the *Parameters-Media* dialogue window. Note that the above instruction will not change the type of the medium. Thus for example if the considered medium was declared in the *Parameters-Media* dialogue (or in **INSERTMEDIUM** command) as isotropic, the values of epsy, epsz, muy, muz etc. declared as arguments of **MEDIUMPAR** will be ignored. This command allows using material permittivity, permeability, etc. as UDO parameters. Thus it allows for example changing medium parameters in a quasi-continuous way, as variables modified in a loop. The names of parameters (see [Fig.E 2.5.3-1](#)).
- **THERMALPAR**(<mediumname>,<ini_temp>,<spec_heat>,<therm_cond_X>,<therm_cond_Y>,<therm_cond_Z>) – analogous to **MEDIUMPAR**, but sets thermal parameters: initial temperature, specific heat, and thermal conductivity in the three coordinate directions.
- **MEDIUMCOL**(<mediumname>, <pen_R>, <pen_G>, <pen_B>, <pen_style>, <pen_width>, <brush_pen_R>, <brush_pen_G>, <brush_pen_B>, <brush_bkg_R>, <brush_bkg_G>, <brush_bkg_B>, <brush_style>) - Defines the pen and brush attributes associated with the medium “mediumname”.
 - <pen_R>, <pen_G>, <pen_B> - RGB parameters of the medium pen line colour
 - <pen_width> - the medium pen line width
 - <pen_tyle> - the medium pen line style (0-continuous, 1- dashed, 2- dotted)
 - <brush_pen_R>, <brush_pen_G>, <brush_pen_B> - RGB parameters of the medium brush pen colour
 - <brush_bkg_R>, <brush_bkg_G>, <brush_bkg_B> - RGB parameters of the medium background brush colour
 - <brush_style> - the medium brush style (hatching style).
- **DISPERSION**(<mediumname>,<model_name>,<par_1>,<par_2>,<par_3>,<par_4>) – sets dispersive dielectric parameters. The *model_name* options are: *DRUDE*, *DEBYE*, *LORENTZ*.
- **SETOBJMED**(<obj_name>,<med_name>) – Sets medium of object *obj_name* to medium *med_name*, if *med_name* exists in *project*.

- **GRIDHV** (<orientation_h>, <orientation_v>) - Declaration of wire grid, allowed only inside the frame of the *element* declared as *WIRE* (see also [Section E 2.2.7](#)).
- **ADJUSTP** (<x>, <y>, <z>, <r>) - Moves all points of the elements at level z close to coordinate (x,y) exactly to (x,y) position. Close to means that points at level z are in the square with corners (x-r,y-r) and (x+r,y+r).
- **XCOORD** (<item_type>, <name>, <n>) - A function returning x-coordinate of n-th point of the element

<item_type> - subject to operation :

ELEM - elements,

ELEML – local elements (those created in the currently open *object*),

<name> - name of the element in *item_type* group (the first found is taken into account)

<n> - the ordinal number of the point (the first one is numbered 0).

- **YCOORD** (<item_type>, <name>, <n>) - The same as **XCOORD** but for y-coordinate.
- **ZCOORD** (<item_type>, <name>, <n>) - The same as **XCOORD** but for z-coordinate.
- **SECTION** (<level>, <height>, <height2>, <medium name>, <name>) - Start of section frame, where:

<level> - level of section *element(s)* base, usually relative to predefined origin variable z,

<height> - height of upper *element*, if equal to zero suppresses +Z activity,

<height2> - height of lower *element*, if equal to zero suppresses -Z activity,

<medium name> - name of element(s) medium (if there is no such medium air is forced),

<name> - max. 20 characters name of element (suffix U/D is added for upper/lower element).

- **ENDSECTION** - End of section frame.

E 3.4. Variables, functions, arithmetic expressions and file reading

The *UDO* language is simple. Here are some general rules:

- Declaration of the variables is automatic and is done by:
 - ♦ Declaration as one of the *UDO* parameters. In a such case it is assumed to be numerical or text depending on the declared initial value.
 - ♦ Appearing on the left side of an equation. In such a case it is assumed to be numerical or text depending on the value of the right side of the equation.
- Strings not declared as variable names and not recognized as key words of *UDO* language are treated as text.
- Function **VAR** (*text*); returns the value of the variable of the name equal to *text*.
- All variables are global within one *UDO*. The values of variables can be transmitted to another *UDO* only through *UDO* parameters.

- Three variables denoting the coordinates of the object origin (x, y, z) are predefined and can be set/modified in a dialogue box.
- Arithmetic expressions and standard functions are similar to those used in C-language.
- It is possible to read numerical variables from a file. To do that we need first to open such a file using the instruction *OPENF*(<file name>). Then *READF* will be returning the consecutive number read from the file. *READF* can be placed at the right side of the equation (e.g. $a=READF$;) as well as inside an equation (e.g. $a=2*\sin(READF)+b$);. Opening of a new file automatically closes the file previously opened (if any).

Numbers and variables:

All the parameters of *Elements (Ports)* and the coordinates used in *NEWLINE*, *ADDLINE* must belong to one of the following categories:

- *Numbers* (or strings where a string is expected as a parameter)
- *Object parameters* as declared in *PAR* declaration
- *Variables* which are declared by their first appearance at the left side of a substitution equation with a number or expression at the right side, like for example “ $k=5$,” or “ $k=2*n+1$ ”. The type of the variable is assigned automatically by the *UDO* interpreter.
- Values of *READF* function
- *Expressions* composed of items belonging to the above categories and the following operators and math functions.

Operators:

++	increment,
--	decrement,
~	change sign
+	addition,
-	subtraction,
*	multiplication,
/	division,
^	power
== != < > >= <= =<	logical conditions,
()	parenthesis,
;	instruction delimiter,
,	arguments separator
=	substitution,
&&	logical and
	logical or
@	string concatenation.

Math functions:

- *sin cos tan asin acos atan atan2* - trigonometric functions and their inverses, **Attention!** arguments of trigonometric functions and results of their inverses are degrees!
- *log logd exp sqrt abs sgn int frac rand srand* - mathematical functions

Other commands and functions:

- **OPENF**(*<file name>*) - Close the file previously open (if any) and open a disk text file for reading. Default file location is in the *project* directory.
- **READF** - Function returning the value of the next number read from the file defined by **OPENF** command.
- **READS** - Function returning the next string read from the file defined by **OPENF** command.
- **VAR** (*<text_expression>*) - It is a function, which returns value of variable given as text expression.
- **STR** (*<arithmetic expression >*) - It is a function converting arithmetic expression (a number) into string.
- **CODE** (*<string>*, *<n>*) - A function returning ASCII code of n-th char in string or -1 if n<0 or n>string length.
- **STL** (*<x1>*, *<y1>*, *<z1>*, *<x2>*, *<y2>*, *<z2>*) - A command drawing STL format section.

Additional remarks

- Note that string concatenation allows definition of quasi-arrays. For example we can introduce a set of variables *a1,a2,a3...* using expressions *a@i*, where *i* is an index variable. However to avoid confusion between a string *ai* and a variable *ai* the following rules need to be respected:
 - Let us assume that the index variable *i* has already been declared and we want to declare a set of variables *ai*. In such a case we should use the instruction:
(a@i)=..... Note that the parenthesis is indispensable here.
 - When the variable *a@i* is already defined and we want to apply it:
 - with a particular value of the index we simply write *a1, a2* etc.
 - with a general index *i* we need to use the function *VAR(a@i)*.
- Let us consider the following modification of *threeel.udo* described in [Section E 3.3](#):

```
comment="modified example of three elements";
bitmap="threeel0.bmp";

PAR("name", oname, "3elm");
PAR("length ", a, 6 );
PAR("width ", b, 5 );
PAR("shape ratio top/bottom", sr, 0.5);
PAR("medium", med, air );
PAR("disk file",df, "c:\temp\points.txt");
ENDHEADER;

TEST( (a>0)&&(b>0), "Dimensions should be positive" );

OPENF(df);
h1=READF;
h2=READF;
h3=READF;
TEST( (h1>0)&&(h2>0)&&(h3>0), "Heights should be positive" );
.. (continuation like in the original threeel.udo)
```

If the file *points.txt* contains the consecutive numbers 5 7 3 the result of drawing the modified *UDO* will be the same as with the original one. Note also that:

- o In the above example full path to the *points.txt* file has been given. Other possibilities of specifying the file location are summarised in [Section E 3.10](#).
- o *READF* function can be also used to overwrite the current values of variables. This opens a way for external modification of the shape of an object drawn using an *UDO*. Such a possibility is actually often used when *QW-3D* is applied together with an external optimiser. It is described in [Section S 3.7](#) of this manual.

E 3.5. Loops in UDO language

When we want to specify an ordered distribution of lines or elements, we need to create a loop structure. In the *UDO* language the “*if*” and “*while*” instructions can be used for this purpose. They have the following syntax:

Syntax of loops:

- **if** <logical expression> **do** **endif;** - Instructions between *do* and *endif* are interpreted if *logical expression* is true.
- **while** <logical expression> **do** ... **endwhile;** - Instructions between *do* and *endwhile* are executed in loop as long as *logical expression* is true.

Additional remarks

There is a very important difference between the equality sign “==” used in logical expressions and the substitution sign “=” used in equations. Using the later one in the logical expression will actually execute the substitution. Thus it will not only cause incorrect loop operation but may change the operation of the rest of the *UDO* script.

Example: taper.udo

The following example will show the use of the *while* loop in generation of a combined element of circular bottom and top.

```
comment="Tapered section";
bitmap="taper.bmp";

PAR("Object name",onam, "taper");
PAR("radius1(r1)",r1,5);
PAR("radius2(r2)",r2,2);
PAR("height(h)",h,10);
PAR("sectors(n)",n,16);
PAR("medium",med,"metal");
ENDHEADER;

TEST(n>2,"n should be greater than 2");
step=360/n;

OPENOBJECT(onam);
ELEMENT(z,0,5,med,Taperb,IN); #bottom
NEWLINE(x+r1,y,x+r1*cos(step),y+r1*sin(step));
i=3;
while i<=n do
  ADDLINE(x+r1*cos((i-1)*step),y+r1*sin((i-1)*step));
  i++;
endwhile;
CLOSELINE;
ENDELEM;
```

```

ELEMENT(z+h,0,6,med,Taperb,IN); #cover
NEWLINE(x+r2,y,x+r2*cos(step),y+r2*sin(step));
i=3;
while i<=n do
  ADDLINE(x+r2*cos((i-1)*step),y+r2*sin((i-1)*step));
  i++;
endwhile;
CLOSELINE;
ENDELEM;
CLOSEOBJ;

```

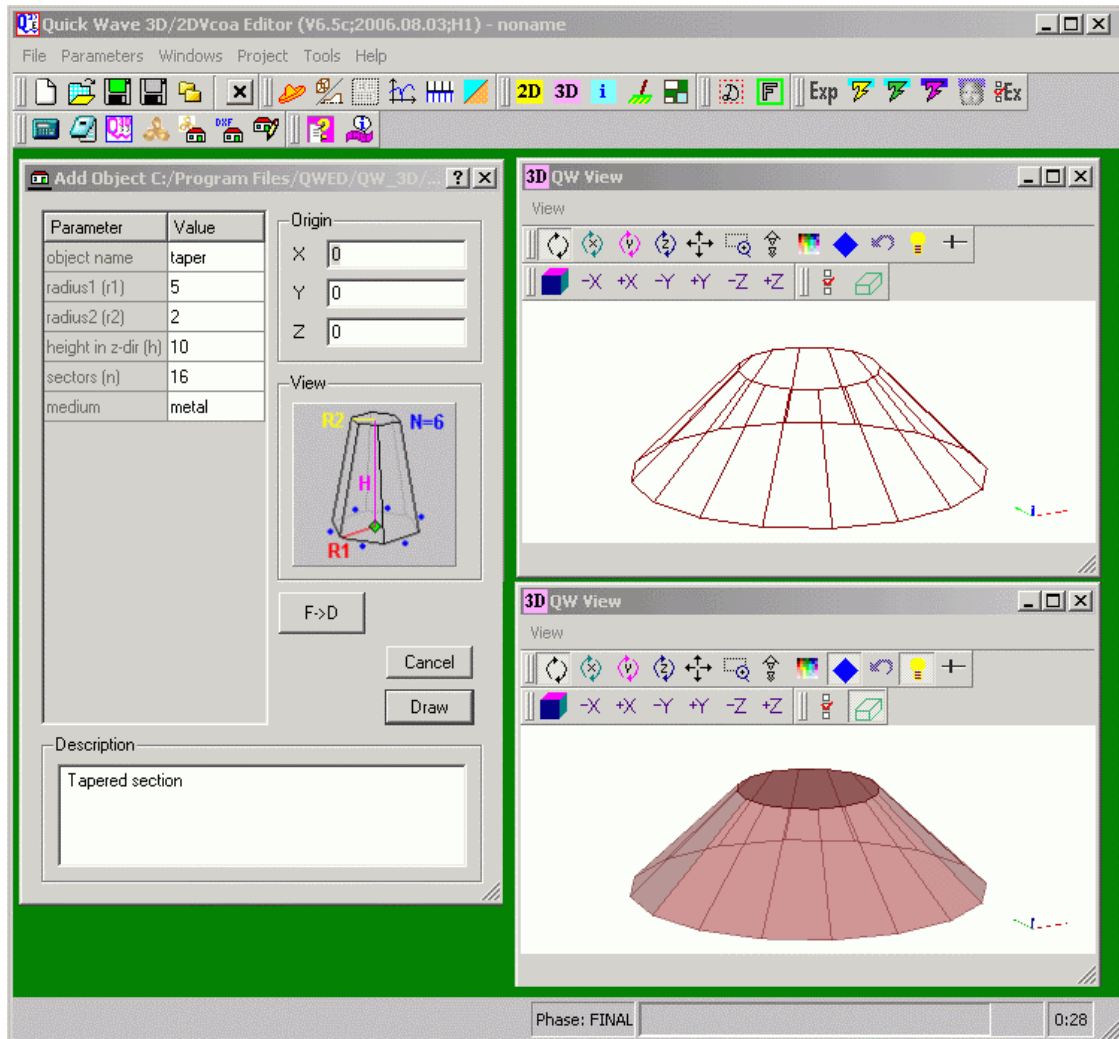


Fig.E 3.5-1 The *Draw-Object* dialogue for *taper.udo*, and 3D views of the produced *object*.

E 3.6. Defining ports, boxes, mesh snapping planes, postprocessing and circuit parameters

We assume that the typical way of introducing ports in UDO scripts is by calling one the port UDOs. They are available in the standard libraries: *ports* (in the form of objects) and *elements* (in the form of elements not assigned to any object).

But it is also possible to introduce ports using directly the primitive *UDO* instructions. In such a case we will proceed in a similar way as when introducing *ports* directly from the *QW-Editor*. Here are the basic hints about it:

- There are two types of *ports*: vertical and horizontal. When drawing a vertical *port* we set its height and draw just one line (*base of element*). Let us look at such a *port* described in *UDO* language:

```
PORT(z,H,INPTEMPLATE,UP,input,rinput);
  NEWLINE(x-W,y,x+W,y);
  NEWLINE(x,y,x,y);
ENDPORT;
```

First we declare the port at the level z having the height H . It is a *source* of the type *transmission line*, which in *UDO* language is described by the declaration of the type *INPTEMPLATE*. Then we declare its activity *UP* meaning, that the energy is supposed to flow in the direction of increasing values of X or Y coordinate. We specify also the name of the *port* and the name of the corresponding *reference*. Next we define the line being the *base of the port*. There is also another line, which follows. This second line is interpreted as a declaration of two points inside the *port*. The first point shows the place of excitation during the *template generation*. The other one is not relevant in most applications. Both points are placed at the height of $H/2$ unless this is changed using *PORTEXC* command.

- Let us now declare a similar *port* but in the XY instead of YZ plane.

```
PORT(z,0,INPTEMPLATE,UP,input,rinput);
  NEWLINE(x-W,y-H/2,x+W,y-H/2);
  ADDLINE(x+W,y+H/2);
  ADDLINE(x-W,y+H/2);
  CLOSELINE;
  NEWLINE(x,y,x,y);
ENDPORT;
```

Here we have the *port height* equal zero and we must declare a rectangle in the plane XY . It is done in the same way as in the case of introducing a rectangular base of an *element*. As in the previous case, we declare here an extra line which in fact is a declaration of two points, the first one being the point of field excitation during the *template generation*.

- In the above examples we have considered a *port*, which is supposed to have a *reference*. The *reference* should be declared as a separate *port* placed immediately after the corresponding *I/O port*. In the case of the first *port* considered in this Section, its *reference* could be declared as follows:

```
PORT(z,H,REFERENCE,UP,rinput,input);
  NEWLINE(x-W,y+d,x+W,y+d);
  NEWLINE(x,y+d,x,y+d);
ENDPORT;
```

The *reference* was declared to appear at a distance d from the *port*. Note that the second *NEWLINE* declaration appears here for the reason of uniformity of the port formats and does not contain relevant information.

- It should be noted here, that *mesh snapping planes* are treated by the software as a kind of dummy *ports*, not performing the input-output functions. Thus, they are declared in the **.udo* file in the similar way as other *ports*, and their type should be declared as *SPECIAL*. For example, to declare a *mesh snapping plane* for $y=d$ we may write:

```

PORT(z,H,SPECIAL,NONE,sp1,NULL);
  NEWLINE(x-W,y+d,x+W,y+d);
  NEWLINE(x,y+d,x,y+d);
ENDPORT;

```

The parameters x , z , H and W influence presentation of the *special plane* in QW-Editor. They should not influence the electromagnetic analysis.

- So far we have considered how the geometry of the *port* can be defined in UDO. It is also possible to specify some parameters of the *port* by including additional commands **inside the port frame** (between the *PORT* and *ENDPORT* commands). There are the following commands available: *PORTPAR*, *PORTPAR2*, *GETIOPAR*, *PORTEXC*, *PORTPOSTP*.

Syntax of port commands:

- *PORT* (<level>, < height >, <type>, <activity>, <name>, <reference>) - Starts the port frame.

<level> - level of port base,

<height>- height of port,

<type> - type of port (corresponding port types of QW-Editor shown in brackets)

<i>POINT</i>	(Lumped source/probe, source)
<i>PROBE</i>	(Lumped source/probe, load)
<i>INPTemplate</i>	(Transmission line, source)
<i>OUTTemplate</i>	(Transmission line, load)
<i>MUR</i>	(Absorbing wall)
<i>PML</i>	(Perfect matching layer)
<i>SHORT</i>	(Symmetry plane, electric)
<i>OPEN</i>	(Symmetry plane, magnetic)
<i>SPECIAL</i>	(Mesh snapping plane)
<i>NEUTRAL</i>	(Neutral mesh snapping plane)
<i>BETWEEN</i>	(Suspended mesh snapping plane)
<i>CONNECT</i>	(Connections between subcircuits)
<i>SUBCBASE</i>	(Subcircuit box)
<i>REFERENCE</i>	(Reference plane)
<i>NEAR2FAR</i>	(NTFbox)
<i>PLANEWAVE</i>	(Plane wave box)
<i>CONTOUR_E</i>	(Electric contour)
<i>CONTOUR_H</i>	(Magnetic contour)
<i>FDMONITOR</i>	(FD monitor)
<i>SUBGRID</i>	(Subgridding box)
<i>SUBREGBORD</i>	(Subcircuits border)
<i>LUMPEDIMP</i>	(Lumped impedance)

<activity> - activity direction of port:

UP - positive direction of axis,

DOWN - negative direction of axis,

NONE - port without activity.

<name> - max. 7 characters name of port,

<reference> - max. 7 characters name of the associated port (*reference* for I/O port, I/O port for *reference*, **NULL** for I/O ports without *reference* planes).

- **PORTPAR**(<size->, <size+>, <activity->, <activity+>, <port_plane>, <port_circumf.>) - Sets mesh snapping and mesh enforcing properties of the port, similarly as the *Advanced features* dialogue called from the I/O ports dialogue

<size-> - maximum cell size in the negative direction starting from the port plane

<size+> - maximum cell size in the positive direction starting from the port plane

<activity-> - 0 or 1; if 1 <size-> is active

<activity+> - 0 or 1; if 1 <size+> is active

<port_plane> - 0 or 1; if 1 mesh is snapped to the port plane

<port_circumf.> - 0 or 1; if 1 mesh is snapped to the port circumference.

- **PORTPAR2**(<subcircuits_margin>, <active_walls>, <dummy>, <dummy>) - Sets *subcircuits_margin* property of the port, similarly as the *Advanced features* dialogue called from the I/O ports dialogue

<subcircuits_margin> - adjacent subcircuits overlay (number of common cells)

<active_walls> 3DP PLANEWAVE active walls coded on bits:

32:X-, 16:X+, 8:Y-, 4:Y+, 2:Z-, 1:Z+ (parameter ignored for -1)

<dummy> - parameter for future usage (value -1 recommended)

- **GETIOPAR**(<file_name.iop>) - The I/O port parameters are read from the file *file_name.iop*. For principles of the *.iop file location please refer to [Section E 3.10](#).
- **PORTEXC**(<point1z>, <point2z>) - Allows to move the excitation points to arbitrary z-level. It is needed to use special excitation point arrangement for template calculation in vertical ports.
- **PORTPOSTP** (<arg1>, <arg2>) - Refers to *lumped source/probe* and determines in which *postprocessing* it is to be included (analogous to *FD-probing* and *[S]* as port No. check boxes in the I/O ports dialogue of QW-Editor)

<arg1> - 0 or 1; if 1 the *lumped source/probe* is included in *FD-probing*

<arg2> - 0 or 1; if 1 the *lumped source/probe* is included in S-parameter extraction

- **EPSEFF** (<value>) - Sets port effective permittivity
- **LUMPED** (<component>, <options>, <resistance>) – Sets parameters for lumped (source/probe) port.

<component> - index 0 to 5 for Ex, Ey, Ez, Hx, Hy, Hz

<options> - for 0 <resistance> is used, 1 denotes INF resistance, 2 is self adjust option

- **LUMPIMPPAR** (<type., component>, <Rp>, <Lp>, <Cp>, <Rs>, <Ls>, <Cs>) – Sets parameters for lumped impedance.

<type> - index 1 to 5 for Parallel, Series, Drude, Debye, Lorentz

<component> - index 0 to 2 for Ex, Ey, Ez

<Rp>, <Lp>, <Cp>, <Rs>, <Ls>, <Cs> - equivalent circuit parameters

- **MODE** (<mode_index>) – Sets port mode (1 - *Quasistatic*, 8 - *multiTEM*, 9 - *planeTEM*, otherwise *Dynamic*)
- **TEMPD** (<method>, <match_freq>, <within>, <from>, <to>, <step>, <component>, <iterations>) – Sets parameters for port dynamic template accordingly to I/O Port dialogue controls
 <method> - 0/1 automatic/manual
- **MTEM** (<medium_name>, <potential>) – Assigns potential to metal materials in port
- **PLWAVE** (<phi>, <theta>, <polarisation>) - Sets plane wave angles.
- **PLWBEAM** (<beam_type>, <neck_origin_X>, <neck_origin_Y>, <neck_origin_Z>, <neck_diameter>, <angle_of_variation>) - Sets beam parameters.
- **PROFILE** (<type>, <par_a>, <par_b>, <file_name>) – Sets PML profile parameters.
 <type> - 0,1,2
- **SYMMETRY** (<symm_v>, <symm_h>) – Defines port vertical and/or horizontal symmetry(-ies)
 <symm_v>, <symm_h> - 0 or 1; if 1 the symmetry is on
- **WAVEFORM** (<shape>, <f1>, <f2>, <duration>, <amplitude>, <delay>, <file_name>) – Defines excitation parameters accordingly to I/O port dialogue
 -1, 0, ..., 7 for no_excitation, delta, ..., defined by user text file
- **WAVEFORM2** (<amplitude_im>, <delay_im>) – Defines imaginary parts of excitation parameters for periodic circuits accordingly to I/O port dialogue.
- **ENDPORT** - End of the port frame.

Here is the review of the basic points about port definitions in UDO:

- There are two types of *ports*: vertical and horizontal.
- A vertical *port* consists of one line (*base of element*) and has non-zero height.
- A horizontal *port* consists of four lines (closed rectangle) and its height is zero.
- Each *port* has an additional line. The first point of this line determines excitation/probe points of the *port*. Even if such a line is dummy, it has to be placed at the end of the *ports* frame. For vertical *ports* this additional line is shifted by the interpreter to the half of the *height of port*.
- *Reference port* should be placed immediately after the corresponding I/O *port*.
- Note that modifying an *object* really means cancelling it and recreating. That is why during this operation all the *port* parameters are also cancelled and set to default values. In practical application of *objects*, this would cause some inconvenience which may be avoided the following way: Before the modification of an *object* including *ports*, the parameters of each of the *ports* can be saved on disk using *Put* command in the I/O *Ports* dialogue. Then, after modification of the *object*, they can be restored using *Get* command in the same dialogue. But the best solution is to execute the *Get* command automatically from *UDO*, using the *GETIOPAR* command.

Example: The library object portox.udo

```
comment="Port for transmission in X-dir with excitation in the port centre";
bitmap="portx1.bmp";

PAR("port name",pname,portox);
PAR("port height (Z-direction)",hpo,3);
```

```

PAR("port width (X-direction)",wip,3);
PAR("port-to-circuit dir. (UP or DOWN)",pdir,"UP");
PAR("1-inp.,2-outp.",kind,1);
PAR("port-ref. distance",ptr,3);
PAR("port IOP file",pio,NO);
ENDHEADER;

TEST((kind==1) || (kind==2),"port can be only 1-input or 2-output");
TEST((pdir==UP) || (pdir==DOWN),"energy transmission can be only UP or DOWN");
y1=0.5*wip;

OPENOBJECT(pname);
portkind=INPTEMPLATE;
actref=pdir;
if (kind==2) do
    portkind=OUTTEMPLATE;
    if pdir=="UP" do actref="DOWN";endif;
    if pdir=="DOWN" do actref="UP";endif;
endif;
refpos=ptr;
if pdir=="DOWN" do refpos=-ptr;endif;
portkind=INPTEMPLATE;
if (kind==2) do
    portkind=OUTTEMPLATE;
endif;

PORT(z,hpo,portkind,pdir,pname,r@pname);
NEWLINE(x,y,x,y+wip);
NEWLINE(x,y+y1,x,y+y1);
if pio!="NO" do GETIOPAR(pio@".iop");endif;
ENDPORT;

PORT(z,hpo,REFERENCE,actref,r@pname,pname);
NEWLINE(x+refpos,y,x+refpos,y+wip);
NEWLINE(x+refpos,y+y1,x+refpos,y+y1);
ENDPORT;
CLOSEOBJ;

```

Postprocessing and circuit parameters:

- **FDPROB** (<lower_freq>,<upper_freq>,<freq_step>) – Enables frequency domain probing
- **FDPROBPAV** (<lower_freq>,<upper_freq>,<freq_step>) – Enables frequency domain probing for power available from the source
- **SK1DIFF** (<lower_freq>,<upper_freq>,<freq_step>,<assumptions>) – Activates SK1 differential postprocessing (S-correction *assumptions* options: 0 - nonreciprocal N-port, 1 - reciprocal N-port, 2 - reciprocal lossless 2-port)
- **SMNDIFF** (<lower_freq>,<upper_freq>,<freq_step>,<assumptions>,<mode>,<IterForS>) – Activates Smn differential postprocessing (S-correction *assumptions* options: 0 - nonreciprocal N-port, 1 - reciprocal N-port, 2 - reciprocal lossless 2-port; *mode* options: 0 - single simulator & single thread, 1 - multisimulator, 2 - multithread)
- **NTF** (<frequencies_string>) – Enables NTF postprocessing
- **NTFBKG** (<eps>,<mu>,<sig>,<msig>) – Defines NTF background dielectric

- **ALLOWBHM** – Enables *BHM* module
- **ALLOWHF** – Enables *Heat Flow* (heat transfer analysis in *BHM* module)
- **PRONY** (*<take_every>*, *<below>*) – Enables Prony postprocessing (parameters like in the postprocessing dialogue)
- **EVS** (*<solutions>*, *<f0>*, *<iterations>*) – Enables eigen value solver
 - <solutions>* - number of searched eigensolutions
 - <f0>* - central frequency
 - <iterations>* - number of simulator iterations
- **CIRTYPE** (*<type>*, *<default medium>*) – Sets circuit type (*type* options: 2 - 3DP, 4 – 2DVcoa, otherwise 3D) and default material (metal or air)
- **PERIODIC** (*<activity_x>*, *<activity_y>*, *<activity_z>*, *<beta_x>*, *<beta_y>*, *<beta_z>*) – Defines features for periodic circuits
- **UNITS** (*<space>*, *<frequency>*) – Sets *project* units (*space* options: 0 - millimeters, 1 – micrometers, 2 - inches, 3 – mils, 4 – meters, 5 - nanometers; *frequency* always GHz)
- **ANGVAR** (*<n>*) – For 2DVcoa circuits sets modal angular variation (range [0-99]). For 3DP circuits sets phase shift per period
- **LOSSES** (*<control_string>*) – Controlling losses suppression and metal loss bandwidth. The following substrings are recognized in *control_string*
 - "-M" - suppress magnetic losses
 - "+M" - enable magnetic losses
 - "-E"/"+E" - suppress/enable electric losses
 - "-P"/"+P" - suppress/enable metal losses
 - "BN"/"BD"/"BT" - narrow/decade/two-decades metal losses bandwidth.

E 3.7. Moving the elements, copying them and controlling their intersections

So far we have explained how we can use the *UDO* language to define shapes. However, it would be difficult to describe some commonly used shapes this way. For example, let us imagine a cross-junction of two coaxial lines. Although the lines are of simple cylindrical shape, the analytical description of their intersection would be complicated. Moreover, there are problems with defining such a junction as a result of logical operations on the two coax lines. Note that no single topological operation performed on the volume occupied by dielectric, or on the volume occupied by metal, produces the desired result. We can obtain this result from simple cylindrical shapes only by performing several consecutive operations like: joining together the inside connectors, joining together the dielectric fillings and then placing the obtained combined connector inside the combined dielectric structure. This is how we would proceed to create such a structure in the *QW-Editor*.

QW-ObjectGenerator is prepared to imitate this process following commands stored in the **.udo* file. We have a broad choice of commands available here. In general, we wanted to make it possible to execute all the commands available in the *QW-Editor*. As a result, we can create structures of complicated shapes, and also observe the consecutive phases of the creation, even in the form of animation. Example at the end of this section shows a process of making a hole in a block of teflon.

We define a block and a tool. Then we move the tool into the block, make the hole using JOIN operation, and we delete the tool.

The syntax of the UDO language commands related to moving, copying and joining elements or objects:

- **MARKFJ** (<item_type>,< range >,<command>) – This command is used to *Mark* or *Unmark* elements or objects as *Passive* or *Active* for subsequent JOIN operation.

<item_type> – subject to operation:

ELEM – elements,

ELEML – local elements (those created in the currently open object),

OBJECT – objects,

OBJECTL – local objects (those created in the currently open object),

<range> – range of operation:

name – name of a single element or object to be marked,

ALL – all elements or local elements or objects or local objects,

ALLACTIVE – all active elements or local elements,

ALLPASSIVE – all passive elements or local elements,

LAST – last created element or object

<command> – operation:

ACTIVE – mark for join as active element or object,

PASSIVE – mark for join as passive element or object,

RESET – unmark elements or objects

- **MARK** (<item_type>,<range>,<command>); – This commands is used to *Mark* or *Unmark* elements or objects for further operations of movement or reproduction.

<item_type> – subject to operation:

ELEM – elements,

ELEML – local elements (those created in the currently open object),

OBJECT – objects,

OBJECTL – local objects (those created in the currently open object),

<range> – range of operation :

name – name of a single element or object to be marked,

ALL – all elements or local elements or objects or local objects,

ALLACTIVE – all active elements or local elements,

ALLPASSIVE – all passive elements or local elements,

LAST – last created element or object

<command> – operation:

SET – mark element or object,

RESET – unmark elements or objects

- **JOIN** (<operation>) - Joining elements according to the rules explained in [Section UG 2.12.1](#).

<operation> - type of join operation,

CUT**INTERSECT****GLUE**

- **SHIFTM** (<dx>,<dy>,<dz>) - Shifting all the marked elements or objects.
 <dx>,<dy>,<dz> - requested shift in the direction of the main axes.
- **MIRROR** (<xy>,<xz>,<yz>) - Performing a mirror reflection of all the marked *elements* or *objects* with respect to the planes $z=0$ or/and $y=0$ or/and $x=0$.
 <xy>, <xz>,<yz> - Boolean variables (0 or 1) defining if the reflection with respect to the particular plane is to be performed.
- **MIRRORX** (<xmirror>) - Performing a mirror reflection of all the marked *elements* or *objects* with respect to the plane $x = xmirror$.
- **MIRRORY** (<ymirror>) - Performing a mirror reflection of all the marked *elements* or *objects* with respect to the plane $y = ymirror$.
- **MIRRORZ** (<zmirror>) - Performing a mirror reflection of all the marked *elements* or *objects* with respect to the plane $z = zmirror$.
- **ROTATE** (<angle>,<x0>,<y0>) - Reflecting all the marked *elements* or *objects* with respect to the specified axis of symmetry, by a specified angle.
 <angle> - angle of rotation
 <x0>,<y0> - coordinates of the axis of symmetry
- **COPY** (<xx>) - Copying the marked *element*. Note that we obtain a second *element* of the same name and situated in the same place. To distinguish the second element from the original one, we can use for example the command **RENAME** or **MARK** with the <range>=LAST.
 <xx> - a dummy numeric variable irrelevant in the present version.
- **REDRAW** (<delay>) - Redrawing all the elements on the screen with a specified delay. When <delay> > 0 the REDRAW operation is performed before the delay, when <delay> < 0 then its absolute value is taken as the actual delay, and the REDRAW operation is performed after the delay.
 <delay>- is the delay time in seconds.
- **RENAME** (<item_type>,<range>,<new_name>) - Changing the name of an element or an object.
 <item_type> - subject to operation:
 ELEM - element,
 OBJECT - object
 <range> - range of operation:
 name - name of a single element or object to be marked,
 LAST - last created element or object
 <new_name> - string representing the new name of the element or the object.
- **DELETE** (<item_type>,<range>) - Deleting element(s) or object(s).
 <item_type> - subject to operation:
 ELEM - elements,
 ELEML – local elements (those created in the currently open *object*),

OBJECT – objects,

OBJECTL – local objects (those created in the currently open object),

<range> - range of operation:

name - name of a single element or object to be marked,

ALL - all elements or local elements or objects or local objects,

ALLACTIVE - all active elements or local elements,

ALLPASSIVE - all passive elements or local elements,

LAST - last created element or object

- **SETPEN** (<colour_R>, <colour_G>, <colour_B>, <width>, <style>) - Defines the pen colour, line width and style for drawing the elements marked with the MARK command.

<colour_R>, <colour_G>, <colour_B> - RGB parameters of the line colour

<width> - line width

<style> - line style (0-continuous, 1- dashed, 2- dotted)

Example : A tool interacting with a block (tool.udo)

UDO script for QW-3D

```
comment="tool acting with a block";
bitmap="tool.bmp";

PAR("Object name",oname,"toolex");
PAR("Block length",a,12);
PAR("Block width",b,10);
PAR("Block height",h,7);
PAR("Medium",med,"teflon");
PAR("Tool width",ta,4);
PAR("Tool length",tb,5);
PAR("Tool height",th,3);
PAR("Tool vertical pos.",top,4);
PAR("Tool horizontal pos.",thp,2);
ENDHEADER;

OPENOBJECT(toola);
ELEMENT(z,h,0,med,block,IN);
NEWLINE(x,y,x+a,y);
ADDLINE(x+a,y+b);
ADDLINE(x,y+b);
CLOSELINE;
ENDELEM;

yp=y+thp-b*3;
xp= x+a*0.5;

ELEMENT(z+top,th,0,metal,tool,IN);
NEWLINE(xp,yp,xp+ta,yp);
ADDLINE(xp+ta,yp+tb);
ADDLINE(xp+ta*0.5,yp+tb*2);
ADDLINE(xp,yp+tb);
CLOSELINE;
```

UDO script for QW-V2D

```
comment="Tool acting with a block (V2D version)";
bitmap="tool.bmp";

PAR("Block length",a,12);
PAR("Block width",b,10);
PAR("Medium",med,"teflon");
PAR("Tool width",ta,4);
PAR("Tool length",tb,5);
PAR("Tool horizontal pos.",thp,2);
ENDHEADER;

OPENOBJECT(toola);
ELEMENT(z,1,0,med,block,IN);
NEWLINE(x,y,x+a,y);
ADDLINE(x+a,y+b);
ADDLINE(x,y+b);
CLOSELINE;
ENDELEM;

yp=y+thp-b*3;
xp= x+a*0.5;

ELEMENT(z,1,0,metal,tool,IN);
NEWLINE(xp,yp,xp+ta,yp);
ADDLINE(xp+ta,yp+tb);
ADDLINE(xp+ta*0.5,yp+tb*2);
ADDLINE(xp,yp+tb);
CLOSELINE;
ENDELEM;

REDRAW(3000);
MARK(ELEM,tool,SET);
SHIFTM(0,b*2.5,0);
```

```

ENDELEM;
REDRAW(3000);
MARK(ELEM,tool,SET);
SHIFTM(0,b*2.5,0);
REDRAW(3000);
MARK(ELEM,tool,RESET);
MARKFJ(ELEM,tool,ACTIVE);
MARKFJ(ELEM,block,PASSIVE);
JOIN(CUT);
REDRAW(3000);
DELETE(ELEM,tool);
CLOSEOBJ;

REDRAW(3000);
MARK(ELEM,tool,RESET);
MARKFJ(ELEM,tool,ACTIVE);
MARKFJ(ELEM,block,PASSIVE);
JOIN(CUT);
REDRAW(3000);
DELETE(ELEM,tool);
CLOSEOBJ;

```

Note: Please use the *project ..Standard\Tool\emptyex.pro* to execute *examples/tool.udo* in *QW-3D*. This *project* is empty but has the windows prepared for good visualisation of the executed operations.

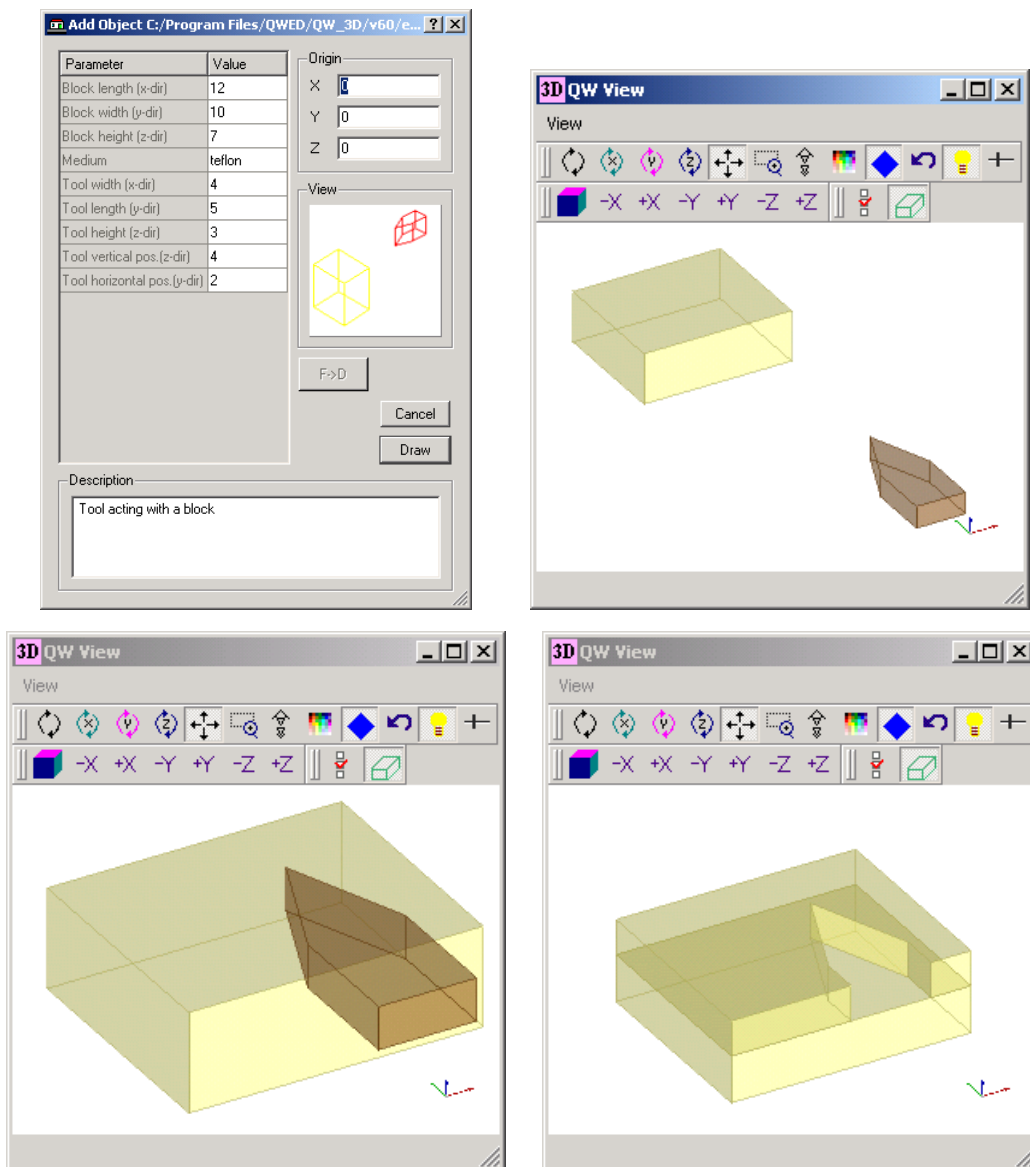


Fig.E 3.7-1 Tool object dialogue (right) and three phases of the object creation as seen in the QW-Editor.

E 3.8. Biphased element and objects (QW-3D only)

Biphased objects are composed of *biphased elements* or/and are drawn differently dependent on the phase (*DRAFT* or *FINAL*) of *QW-Editor*. Let us recall, that in [Section E 3.3](#) we considered three possible types of *monophased elements*:

<type> - type of element: **0** – simple element, **5** - bottom of combined element, **6** - cover of combined element.

Biphase elements may be drawn by declaring one of three other possible types: **10** – simple *biphased element*, **15** - bottom of combined *biphased element*, **16** - cover of combined *biphased element*. Previously used *monophased elements* were enforcing a *sublayer* of the FD-TD grid at its bottom and top. Contrary to that *biphased elements* do not enforce any mesh snapping in the *DRAFT* phase of *QW-Editor* operation. The *DRAFT* phase is used to set the proper distribution of mesh *sublayers*. Then in the *FINAL* phase the *biphased elements* are drawn in a way to fit the mesh.

Syntax of commands used in biphased objects:

- **SLICINGPHASE** - Function returning **0** when *QW-Editor* is in the *DRAFT* phase and **1** when *QW-Editor* is in the *FINAL* phase.

- **MESHINDEX** (<level>) - A function returning:

The **ordinal number** of the *sublayer* of FD-TD mesh situated most closely at or below *level* (if there is an FD-TD mesh at or below level),

-1 (if there is an FD-TD mesh but situated above level),

-2 (if there is no FD-TD mesh).

- **MESH** (<n>) - A function returning the level at which the *n*-th sublayer of the FD-TD mesh is situated. The sublayers are counted up with the lowest sublayer numbered **0**.
- **EXPANDMESH** (<lowerlevel>,<upperlevel>) – Expands mesh limits in *z*-direction between *lowerlevel*, and *upperlevel* to allow direct drawing of *biphased objects* in the *FINAL* phase of the *Editor* without previous drawing in the *DRAFT* phase. It does not work when direct mesh limits are enforced by the user (using *Setup-Mesh-Mesh boundaries*). Attention: side effect of **EXPANDMESH** command is immediate mesh creation.
- **SETSUSPFLAGS** (<draw_suspended>, <skip_EXPANDMESH>,<slicing_phase>) - Sets internal *QW-Editor* flags. Arguments are Boolean values:

<draw_suspended> - 0 or 1; if 1 the next elements will obtain suspended flag

<skip_EXPANDMESH> - 0 or 1; if 1 the **EXPANDMESH** command will be ignored

<slicing_phase> - 0 or 1;

if 1 changes the phase of *QW-Editor* to *FINAL*, (without modifying and redrawing project like  button);

if 0 the *QW-Editor* switches to *DRAFT* phase, (without modifying and redrawing project like  button)

E 3.9. Controlling the mesh

So far it has been explained how to set a *mesh snapping plane* and how to enforce a particular meshing *UP* or/and *DOWN* from such a plane using **PORTPAR** command (see [Section E 3.6](#)). There was also

a presentation (in [Section E 3.8](#)) of the functions *MESH* and *MESHINDEX*, which are very useful in generating *biphased objects*. Here we will supplement these commands with some other commands, which may be useful in controlling the meshing in all directions.

- **MESHPAR** (<arg1>, ..., <arg10>) - Enforces meshing in the whole circuit or in its part.
 - <arg1> - maximum cell size in X direction,
 - <arg2> - maximum cell size in Y direction,
 - <arg3> - maximum cell size in Z direction,
 - <arg4> - lower limit of the meshed zone in X direction,
 - <arg5> - upper limit of the meshed zone in X direction,
 - <arg6> - lower limit of the meshed zone in Y direction,
 - <arg7> - upper limit of the meshed zone in Y direction,
 - <arg8> - lower limit of the meshed zone in Z direction,
 - <arg9> - upper limit of the meshed zone in Z direction,
 - <arg10> - 0 or 1; value 1 corresponds to checked *Adjust to objects* box in *Mesh Parameters* dialogue - it causes that the whole circuit will be meshed and the above parameters <arg4> .. <arg9> will be ignored; value 0 causes that the mesh will be restricted to the region defined by the above parameters <arg4> .. <arg9>.
- **GETMESHPAR** (<n>) - A function returning the *n*-th mesh parameter. Argument *n* has to be in range 1-10 (see *MESHPAR* command description how mesh parameters are enumerated).
- **MESHXINDEX** (<level>) - A function returning:
 - The **ordinal number** of the border between FD-TD cells situated most closely at or below (in $-X$ direction) from *level* (if there is an FD-TD mesh at or below level),
 - 1 (if there is an FD-TD mesh but situated above level),
 - 2 (if there is no FD-TD mesh).
- **MESHX** (<n>) - A function returning the x-coordinate of the *n*-th mesh line (border between FD-TD cells). The mesh lines are counted up (in $+X$ direction) with the leftmost one numbered 0.
- **MESHYINDEX** (<level>) - The same as *MESHXINDEX* but in *Y* direction.
- **MESHY** (<n>) - The same as *MESHX* but in *Y* direction.
- **CREATEMESH** - Forces mesh creation inside UDO file.

Additional comments:

The *MESHPAR* command mimics the *Setup-Mesh* command of *QW-Editor* (see [Section E 2.2.1](#)), and allows a global mesh settings. The mesh can be locally modified with the *PORTPAR* command described in [Section E 3.6](#).

E 3.10. Files used by UDO language and their location

In this Section we shall summarise general principles for locating the various files used in the *UDO* language, and explain how the *UDO* interpreter searches for these files.

***.udo files read from QW-Editor**

When we want to invoke a new *UDO object* from *QW-Editor* via the *Draw-Get-LibItem* command, the contents of the recently used *object library* appear in the dialogue. We can change the directory and pick up the desired *.udo file from any location.

When we want to modify an existing *object* and double-click it on the *Select Object* list, the name of the *.udo file describing this *object* is taken from the *project*. In typical operation of the software, full paths to all the used *.udo files are recorded in the *.pro file, and hence we can be sure that the correct *.udo file will be loaded.

The only case of object modification which is not “typical” occurs if we have previously changed the recorded path to *.udo by text-editing the *.pro. Then the search for the *.udo file follows the same rules as in the case of *.udo called by *CALL* command from *UDO* scripts, as explained below.

***.udo files called by CALL command from UDO scripts**

Let us first establish the meaning of “name” versus “short name” and “full name”. We can consider three basic possibilities of specifying file “name”:

- a) *my_udo.udo*
- b) *c:\dir_1\dir_2\my_udo.udo*
- c) *dir_k\dir_l\my_udo.udo*

In all these cases, the “short name” is the same: *my_udo.udo*. Thus in case a) “name” is also “short name”.

“Name” in case b) is also “full name”.

In general, “name” may include any number of directories up from the file location, and thus may be anything between “short name” and “full name”.

As we will see, the result of *CALL* command depends on how the “name” has been specified. The interpreter tries to locate the file of given “name” in the following sequence of steps:

1. *project* directory + name
2. by standard File-Open command on given name (in practice, works if full name is given)
3. *project* directory + short name
4. in consecutive directories set by the user as *UDO Paths* + name
5. in consecutive directories set by the user as *UDO Paths* + short name
6. in *elib* directory + name

If all six steps are unsuccessful, it terminates with an error message.

***.bmp files substituted for bitmap variable in the header of the UDO program**

When considering a particular *.udo file, the interpreter tries to locate its bitmap file of given “name” in the following sequence of steps:

1. by standard File-Open command on given name (in practice, works if full name is given)
2. *.udo directory + name
3. in consecutive directories set by the user as *UDO Paths* + name
4. in *elib\bitmaps* directory + name

If all four steps are unsuccessful, it terminates with a warning message listing explicitly all paths that have been tried.

***.txt and other files opened by OPENF command (and read from by READF)**

The interpreter tries:

1. *project* directory + name
2. *elib* directory + name
3. by standard File-Open command on given name (in practice, works if full name is given)

If unsuccessful, it terminates with an error message.

***.iop files used by GETIOPAR command**

The interpreter tries:

1. *project* directory + name
2. by standard File-Open command on given name (in practice, works if full name is given)

If unsuccessful, it terminates with a warning message.

E 3.11. Optimising with UDO

Parametric representation of variables in *UDO objects*, combined with command line operation of *QW-Editor*, opens way to automatic circuit design by running *QW-3D* within an external optimisation loop. Interested users may either apply an optional *QW-OptimiserPlus* module or try to couple *QW-3D* with their own optimising procedures. In the latter case, the *project* to be optimised must comply with following rules:

1. The *project* must contain at least one *UDO object*.
2. The *object* must be of the same name as the *project*. This (and only this!) *object* will be reconstructed whenever the *QW-Editor* is called with *-m* parameter (cf. [Section E 2.8](#)).
3. There are two basic possibilities of actually modifying the *object* between successive iterations of the optimisation process:
 - ♦ The first is to directly modify the corresponding **.udo* file. Remember that **.udo* files are text files of strictly defined structure. Therefore this way of operation is possible but highly prone to any operator's mistakes related to finding / replacing appropriate fragments of the **.udo* file.
 - ♦ The other possibility, safer and thus recommended, is to prepare the relevant **.udo* file in such a way that the values of variables taking part in the optimisation process are read (via *READF* function) from **.txt* files. The external optimiser will only need to modify or replace these **.txt* files, which may be short and simple. Note that for this operation to be effective, all variables to be modified in the optimisation process must be declared as *parameters* in the *object header* (via *PAR* function). The *object* can also contain additional *parameters*, not used in the optimisation process.

E 3.12. UDO commands and functions in alphabetic order

| Command | Parameters | Short description |
|-----------------------------------|---|---|
| <u>ADDLINE</u> | (x ₂ , y ₂) | Insert line from previous point to point (x ₂ , y ₂) |
| <u>ADDX</u> | (dx) | Insert line from previous point (x, y) to point (x+dx, y) |
| <u>ADDXYZ</u> | (x, y, z) | Insert contour point |
| <u>ADDY</u> | (dy) | Insert line from previous point (x, y) to point (x, y+dy) |
| <u>ADJUSTP</u> | (x, y, z, r) | Move points at level z to position (x, y) if they are in the square with corners (x-r, y-r) and (x+r, y+r) |
| <u>ALLOWBHM</u> | | Switch on BHM mode |
| <u>ALLOWHF</u> | | Switch on Heat Flow (heat transfer analysis) |
| <u>ANGVAR</u> | (n) | For 2DVcoa circuits sets modal angular variation (range [0-99]). For 3DP circuits sets phase shift per period |
| <u>CALL</u> | (path, par1, par2, ... , parn, parx, pary, parz, N) | Calling another <i>object</i> to be nested as a <i>subobject</i> in the present <i>object</i> |
| <u>CIRTYPE</u> | (type, medium_name) | Sets circuit type and default medium. |
| <u>CLOSELINE</u> | | Insert line from previous point to (x ₁ , y ₁) point of the last <i>NEWLINE</i> |
| <u>CLOSEOBJ</u> | | End of object frame |
| <u>CODE</u> | (string, n) | Function returning ASCII code of n-th char in string |
| <u>COPY</u> | (xx) | Copying the marked <i>element</i> |
| <u>CREATEMESH</u> | | Forces mesh creation inside UDO file |
| <u>DELETE</u> | (item_type, range) | Deleting element(s) or object(s) |
| <u>DIAMETER</u> | (dia) | Declaration of wire diameter, allowed only inside the frame of the <i>element</i> declared as <i>WIRE</i> |
| <u>DISPERSION</u> | (medium_name, model_name, par1, par2, par3, par4) | Declaration of dispersive dielectric data |
| <u>do</u> | | Keyword for if and while commands |
| <u>ELEMENT</u> | (level, height, type, medname, name, spin/wire) | Start of element frame |
| <u>ENDELEM</u> | | End of element frame |
| <u>ENDHEADER</u> | | Separates <i>header</i> and <i>body</i> of the <i>UDO</i> program |
| <u>endif</u> | | End <i>if</i> command |
| <u>ENDPORT</u> | | End of port frame |
| <u>ENDSECTION</u> | | End of section frame |
| <u>endwhile</u> | | Ends <i>while</i> command |
| <u>EPSEFF</u> | (value) | Port effective permittivity setting |

| | | |
|-------------------------------------|---|---|
| <u>EVS</u> | (solutions, f0, iterations) | Enables eigen value solver |
| <u>EXPANDMESH</u> | (lowerlevel, upperlevel) | Expands mesh limits in z-direction |
| <u>FDPROB</u> | (lower_freq, upper_freq, freq_step) | Enables frequency domain probing |
| <u>FDPROBPAV</u> | (lower_freq, upper_freq, freq_step) | Enables frequency domain probing for power available from the source |
| <u>GETIOPAR</u> | (file_name) | The I/O port parameters are read from the file |
| <u>GETMESHPAR</u> | (n) | Function returning the n -th mesh parameter |
| <u>GRIDHV</u> | (orientation_h, orientation_v) | Declaration of wire grid, allowed only inside the frame of the <i>element</i> declared as <i>WIRE</i> |
| <u>if</u> | | Standard control command |
| <u>INSERTMEDIUM</u> | (name, type) | Inserts a medium into the project |
| <u>JOIN</u> | (operation) | <i>CUT, INTERSECT, GLUE</i>
Joining <i>elements</i> |
| <u>LOSSES</u> | (text string) | Controlling losses suppression and metal loss bandwidth |
| <u>LUMPED</u> | (component, options, impedance) | Parameters of lumped port |
| <u>LUMPIMPPAR</u> | (type, component, Rp, Lp, Cp, Rs, Ls, Cs) | Parameters of lumped impedance |
| <u>MARK</u> | (item_type, range, command) | Used to <i>Mark</i> or <i>Unmark elements</i> or <i>objects</i> for further operations of movement or reproduction |
| <u>MARKFJ</u> | (item_type, range, command) | Used to <i>Mark</i> or <i>Unmark elements</i> or <i>objects</i> as <i>Passive</i> or <i>Active</i> for subsequent <i>JOIN</i> operation |
| <u>MEDIUMCOL</u> | (medname, pen_R, pen_G, pen_B, pen_style, pen_width, brush_pen_R, brush_pen_G, brush_pen_B, brush_bkg_R, brush_bkg_G, brush_bkg_B, brush_style) | Material pen/brush colour/style setting |
| <u>MEDIUMPAR</u> | (medname, epsx, mux, sigx, msigx, psy, muy, sigy, msigy, epsz, muz, sigz, msigz, density) | Material parameters setting |
| <u>MESH</u> | (n) | Function returning the z-level of the n -th sublayer of the FD-TD mesh |
| <u>MESHINDEX</u> | (level) | Function returning the ordinal number of the <i>sublayer</i> of FD-TD mesh situated at or below level |
| <u>MESHPAR</u> | (arg1, ..., arg10) | Permits to make global mesh settings |
| <u>MESHX</u> | (n) | Function returning the x-coordinate of the n -th mesh line of the FD-TD mesh. |
| <u>MESHXINDEX</u> | (level) | Function returning the ordinal number of the border between FD-TD cells at or below level (in $-X$ direction) |

| | | |
|-----------------------------------|--|---|
| <u>MESHY</u> | (n) | Function returning the y-coordinate of the n -th mesh line of the FD-TD mesh |
| <u>MESHYINDEX</u> | (level) | Function returning the ordinal number of the border between FD-TD cells at or below level (in $-Y$ direction) |
| <u>MIRROR</u> | (xy, xz, yz) | Performing a mirror reflection of all the marked <i>elements</i> or <i>objects</i> with respect to the planes $z=0$ or/and $y=0$ or/and $x=0$ |
| <u>MIRRORX</u> | (xmirror) | Performing a mirror reflection of all the marked <i>elements</i> or <i>objects</i> with respect to the plane $x=xmirror$ |
| <u>MIRORY</u> | (ymirror) | Performing a mirror reflection of all the marked <i>elements</i> or <i>objects</i> with respect to the plane $y=ymirror$ |
| <u>MIRRORZ</u> | (zmirror) | Performing a mirror reflection of all the marked <i>elements</i> or <i>objects</i> with respect to the plane $z=zmirror$ |
| <u>MODE</u> | (port_mode_index) | Sets port mode |
| <u>MTEM</u> | (medium_name, potential ₂) | Assigns potential to medium in port |
| <u>NEWLINE</u> | (x ₁ , y ₁ , x ₂ , y ₂) | Creates the first segment of a broken line |
| <u>NTF</u> | (frequencies_string) | Defines NTF frequencies |
| <u>NTFBKG</u> | (eps, mu, sig, msig) | Defines NTF background dielectric |
| <u>OPENF</u> | (file name) | Open a disk text file for reading |
| <u>OPENOBJECT</u> | (name) | Start of object frame |
| <u>PAR</u> | (☐ escry.text, var.name, def.Val) | Parameter definition |
| <u>PERIODIC</u> | (activity_x, activity_y, activity_z, beta_x, beta_y, beta_z) | Defines features for periodic circuits |
| <u>PLWAVE</u> | (phi, theta, polarisation) | Defines plane wave data |
| <u>PLWBEAM</u> | (beam_type, neck_origin_X, neck_origin_Y, neck_origin_Z, neck_dia, angle_of_variation) | Defines beam data |
| <u>PORT</u> | (level, height, type, activity, name, reference) | Start of <i>port</i> frame |
| <u>PORTEXC</u> | (point1z, point2z) | Moving the excitation points to arbitrary z-level |
| <u>PORTPAR</u> | (size-, size+, activity-, activity+, port_plane, port_circumf.) | Sets <i>mesh snapping</i> and <i>mesh enforcing</i> properties of the <i>port</i> |
| <u>PORTPAR2</u> | (subcircuits_margin, active_walls, dummy, dummy) | Sets <i>subcircuits_margin</i> property of the <i>port</i> |
| <u>PORTPOSTP</u> | (arg1, arg2) | Refers to <i>lumped source/probe</i> and determines in which <i>postprocessing</i> it is to be included |
| <u>PROFILE</u> | (type, par_a, par_b, file_name) | Defines PML profile |
| <u>PRONY</u> | (take_every, below) | Enables Prony postprocessing |
| <u>READF</u> | | Function returning the value of the next number read from the file defined by <i>OPENF</i> command |

| | | |
|-------------------------------------|--|--|
| <u>READS</u> | | Function returning the next string read from the file defined by <i>OPENF</i> |
| <u>REDRAW</u> | (delay) | Redrawing all the elements on the screen with a specified delay |
| <u>RENAME</u> | (item_type, range, new_name) | Changing the name of an <i>element</i> or an <i>object</i> |
| <u>ROTATE</u> | (angle, x0, y0) | Reflecting all the marked <i>elements</i> or <i>objects</i> with respect to the specified axis of symmetry, by a specified angle |
| <u>SECTION</u> | (level, height, height2, medium_name, element_name) | Start of section frame |
| <u>SETATTR</u> | (which_attribute, attr_value_1, attr_value2) | Sets a property for element |
| <u>SETOBJMED</u> | (obj_name, med_name) | Change object material |
| <u>SETPEN</u> | (colour_R, colour_G, colour_B, width, style) | Defines the pen colour, line width and style for drawing the elements marked with the <i>MARK</i> command |
| <u>SETSUSPFLAGS</u> | (draw_suspended, skip_EXPANDMESH, slicing_phase) | Sets internal <i>QW-Editor</i> flags |
| <u>SHIFTM</u> | (dx, dy, dz) | Shifting all the marked <i>elements</i> or <i>objects</i> |
| <u>SK1DIFF</u> | (low_freq, upper_freq, freq_step, assumptions) | Activates SK1 differential postprocessing |
| <u>SLICINGPHASE</u> | | Function returning 0 when <i>QW-Editor</i> is in the <i>DRAFT</i> phase and 1 when <i>QW-Editor</i> is in the <i>FINAL</i> phase |
| <u>SMNDIFF</u> | (low_freq, upper_freq, freq_step, assumptions, mode, IterForS) | Activates Smn differential postprocessing |
| <u>STL</u> | (x1, y1, z1, x2, y2, z2) | Inserts STL format section |
| <u>STR</u> | (arithmetic_expression) | Function converting arithmetic expression into string. |
| <u>SYMMETRY</u> | (symm_v, symm_h) | Defines port symmetry |
| <u>TEMPD</u> | (method, match_freq, within, from, to, step, component, iter) | Defines dynamic template data |
| <u>TEST</u> | (logical expression, text) | Test and validate expression |
| <u>THERMALPAR</u> | (med_name, ini_temp, spec_heat, therm_cond_X, Y, Z) | Material thermal parameters setting |
| <u>UNITS</u> | (space, freq) | Defines project units |
| <u>VAR</u> | (text_expression) | Function returning value of the variable given as the text expression |
| <u>WAVEFORM</u> | (shape, f1, f2, duration, amplitude, delay, file_name) | Defines waveform parameters |
| <u>WAVEFORM2</u> | (amplitude_im, delay_im) | Defines additional waveform parameters for periodic circuits |
| <u>while</u> | | Standard control command |
| <u>XCOORD</u> | (item_type, name, n) | Function returning x-coordinate of n-th point of the element |

| | | |
|-------------------------------|--------------------|--|
| <u>YCOORD</u> | (item_type,name,n) | Function returning y-coordinate of n-th point of the element |
| <u>ZCOORD</u> | (item_type,name,n) | Function returning z-coordinate of n-th point of the element |

Operators:

| | | |
|-----------------------|------------------------|--|
| ++ | increment, | |
| -- | decrement, | |
| ~ | change sign | |
| + | addition, | |
| - | subtraction, | |
| * | multiplication, | |
| / | division, | |
| ^ | power | |
| == != < > >= => <= =< | logical conditions, | |
| () | parenthesis, | |
| ; | instruction delimiter, | |
| , | arguments separator | |
| = | substitution, | |
| && | logical and | |
| | logical or | |
| @ | string concatenation. | |

Math functions:

sin cos tan asin acos atan atan2 – trigonometric functions and their inverses,

Attention! Arguments of trigonometric functions and results of their inverses are degrees!

log logd exp sqrt abs sgn int frac rand srand – mathematical function.

E 4. Files created or used by QW-Editor

During its operation QW-3D creates and uses a variety of files which can be grouped into five categories:

1. files used by QW-Editor, created by QW-Editor or externally,
2. files created by QW-Editor and used by QW-Simulator,
3. files created and used by QW-Simulator,
4. files created by QW-Simulator for external use,
5. externally prepared files used by QW-Simulator.

In this Section, brief overview of each of the above five categories will be presented. The objective is to give the user some notion about the contents of the files that will appear in his directories, about the possibilities to influence the graphical environment, and about the kind of output data available for external use. Let us also stress that to the authors of the program, the files of the second category, i.e. the communication files between QW-Editor and QW-Simulator, provide powerful means for software debugging. Since all these files are in text format, the user will be able to assist the authors in quick removal of any possible difficulties in software operation.

E 4.1. Files used by QW-Editor, created by QW-Editor or externally

Internal files of QW-Editor comprise *project* files and *project-independent* files.

To discuss the *project-independent* files, let us refer to the directories of installation CD which should be duplicated on the user's computer (see [Section UG 1.4](#)). Let us take a look at the new system of subdirectories concerning QW-Editor.

- *qwbin* contains the actual executable file of QW-Editor *qed.exe*
- *envir* contains files for setting up the environment:
 - *zednqw.ini* - contains various sections which can be modified by the user. Example of this file, as included on the installation CD, has been shown in [Section UG 1.4](#). QW-Editor makes an attempt to read this file whenever it is started. Although QW-Editor will start even if the file is not found, some features of QW-3D may not work in such a case (e.g., *File-Export and Run*). The sections of *zednqw.ini* file are:
 - ♦ [tools] - contains paths to external tools (see remarks on installation in [Section UG 1.4](#)).
 - ♦ [FreqRange] is a default frequency range for the dialogues of *Parameters- I/O Ports* and *Parameters-Postprocessing*.
 - ♦ [IOPortsPar] are default *Template Parameters*.
 - ♦ [DefaultColours] are RGB values of background colours for 2D and 3D windows.
 - ♦ [UndoParams] specifies the number of recent operations that are stored in memory, and thus can be revoked via *Undo* command of QW-Editor. Note that if this section of *zednqw.ini* file does not exist, QW-Editor will set the *UndoListMaxLength* parameter to zero, and the *Undo* command will not be operational!
 - ♦ [Prony] are default parameters for *QProny* module
 - ♦ [EVS] are default parameters for *Eigenvalue Solver*
 - ♦ [UDOPath] specifies one or more additional paths (in which case the number of additional paths is given) for UDO search

- ♦ [MeshParams] specifies the number of cell descriptors that will be allocated for mesh generation. In principle, this number is dynamically re-allocated when needed. However, memory management is not perfect in computer systems and for large problems it may happen that multiple “holes” are left in memory. Then *QW-Editor* will then not be able to allocate a single sufficiently large memory block, even though the total amount of RAM is sufficient for the *project*. If this happens, a warning is issued by *QW-Editor*, and the user may cure memory management problem by increasing the number of pre-allocated descriptors in this *ini* section..
- *zedglob.env* - contains a history of files recently loaded into the editor and the last set dimensions of the main window. This file may exist or not when first starting the *QW-Editor*. It is created / updated upon each exit.
- *media* contains media library files **.mlb*. At least one library *default.mlb* must exist, and it is the first one always considered by the *QW-Editor*. The user can then switch to other libraries from within the *Parameters-Media* dialogue box.
- *elib* contains **.udo* files with description of *external objects* (see [Chapter E 3](#)). Its *bitmaps* subdirectory contains **.bmp* files with bitmaps of corresponding external objects. If an appropriate **.bmp* file is not found, nothing happens except that the *object* is not visualised.

Let us now proceed to *project* files of *QW-Editor*. As explained in [Chapter E 2](#), the *QW-Editor* allows to define geometry of the circuits to be analysed as well as parameters of the analysis. All these data for each *project* are stored in *project_name.pro* file.

To limit the risk of losing important files in the case of hardware or system problems, a back-up file of the *project* file is created upon a *Save* operation, under the name *project_name.pr1*. Upon a successive *Save* operation, *project_name.pr1* is renamed to *project_name.pr2*.

Let us recall that a short description of each *project* is introduced by the user in the *Description* line of the *Parameters-Circuit type* dialogue. This short description is naturally stored in the *project_name.pro* file. However, some users may wish to maintain a more detailed description about their *projects*. While in *QW-Editor*, please press the *T* button of the main toolbar activating the external text editor. A text file *project_name.txt* will be created in the directory containing *project_name.pro*, and later accessible through this shortcut.

Additionally, the user can save the I/O ports parameters in **.iop* files. The user is advised to create *QW_PROJ* directory and to keep the **.pro* files in appropriate subdirectories.

E 4.2. Files created by QW-Editor and used by QW-Simulator

Communication between *QW-Editor* and *QW-Simulator* is accomplished via text files. All these files are created by *QW-Editor* in response to *File-Export* or *File-Export&Run* commands. Most generally, the files can be divided into three types:

- ♦ *tasker files* **.ta3*
- ♦ *parameter files* **.pa3*
- ♦ *shape files* **.sh3*

The actual number and names of files created depend on the problem to be solved. In the simplest case, at least one **.ta3* file, one **.pa3* file, and one **.sh3* file will be created. Syntax of **.pa3* files can easily be deduced from examples since the parameters are strongly associated with the parameters specified by the user in *QW-Editor*, and additional explanations are included in each line following the ‘*’ sign. Thus we shall only formulate a few general remarks regarding the keywords of *tasker files* and basic structure of *shape files*.

E 4.2.1. Basic structure of shape files

A typical structure of an *.sh3 file is presented below:

HEADER (*name, version*)

NO_OF_MEDIA

 {for *i=1..NO OF MEDIA*

 MEDIUM_TYPE, MEDIUM_NAME

list of constitutive parameters appropriate for given MEDIUM_TYPE

 }end of media description loop

STANDARD_CELL_SIZE

NO_OF_SUBCIRCUITS

NO_OF_PORTS

NO_OF_MESH_POINTS_ALONG_x, ...POINTS_ALONG_y, ...SUBPOINTS_ALONG_z

 {for *i=1..NO_OF_SUBCIRCUITS*

list of types of 6 border walls

list of x-coordinates of mesh points along x

list of y-coordinates of mesh points along y

list of z coordinates of mesh subpoints along z

 NO_OF_CELL_DESCRIPTORs

list of cell descriptors

 NO_OF_DIFFERENT_SUBLAYERS

 {for *j=1..NO_OF_DIFFERENT_SUBLAYERS*

 NO_OF_SUBLAYERS_OF_TYPE_j

list of cells in sublayer of type j

 }end of sublayer description loop

 }end of subcircuit description loop

E 5. Index

- 2D windows, 18, 20
- 3D windows, 42, 44
- active library, 63, 68
- AMIGO, 9
- AMIGO Circuit group of commands, 9
- AMIGO FDTD iterations group of commands, 11
- AMIGO Frequency group of commands, 9
- AMIGO Mesh Control group of commands, 10
- antenna, 6, 46
 - lossless, 6
- Automatic Meshing Intelligent Generation Option, 9
- buttons, 17, 38
 - "house", 68
 - <-<-<-, 68
 - >, 68
- Background Colour, 43
- Blend, 43
- Draw SPlanes/boundaries, 39
- Draw-Clear, 39
- Draw-Element-Circle/polygon, 39
- Draw-Element-Rectangle, 39
- Draw-Get-LibItem, 39
- Draw-IOPort, 39
- Edit, 32
- Edit-Element, 39
- Edit-Join-Cut, 39
- Edit-Join-Glue, 39
- Edit-Join-Intersect, 39
- Edit-Line, 39
- Edit-Object, 39
- Edit-Point, 32, 39
- Edit-Port, 39
- Edit-Redo, 39
- Edit-Select Element, 39
- Edit-Select Object, 39
- Edit-Undo, 39
- File-Exit, 17
- File-Export, 17
- File-Export Options, 17
- File-Export&Run, 17
- File-Export&Test, 17
- File-Export, Run&Start, 17
- File-Export, Run&Start Prony, 17
- File-Load, 17
- File-New, 17
- File-Save, 17
- File-Save as, 17
- File-UDOPath, 17
- Fill for Image, 43
- Free Rotation, 43
 - from +X, 43
 - from +Y, 43
 - from +Z, 43
 - from -X, 43
 - from -Y, 43
 - from -Z, 43
- Help-About..., 17
- Help-Contents..., 17
- Hide Ports&Boxes, 43
- HV Grid, 32
- Info-Mesh, 39
- Initial View, 43
- Lights, 43
- Parameters-Amigo, 17
- Parameters-Circuit type, 17
- Parameters-I/O Ports, 17
- Parameters-Media, 17
- Parameters-Units, 17
- Project- Draft Modify All, 17
- Project- Final Modify All, 17
- Setup-Axes, 39
- Setup-Circuit type, 39
- Setup-Grid/Snap, 39
- Setup-Level, 39
- Setup-Mesh, 39
- Setup-Plane, 39
- T, 105
- Tools-ACIS Viewer, 17
- Tools-Calculator, 17
- Tools-DXF Converter, 17
- Tools-SAT Filter, 17
- Tools-Simulator, 17
- Tools-UDO Editor, 17
- Translate, 43
- Undo, 43
- Up/Down Zoom, 43
- View Elements, 43
- View-Extens, 39
- View-Options, 39
- View-Zoom, 39
- Windows-Arrange, 17
- Windows-Open- ProjectInfo, 17
- Windows-Open-2D, 17
- Windows-Open-3D, 17
- Windows-Refresh, 17
- X Rotation, 43
- Y Rotation, 43
- Z Rotation, 43
- cell size, 5, 25, 26, 27, 38
- circuit type, 8, 21, 22
- commands, 7
 - Activity, 50
 - Arrange, 13
 - Axes, 23
 - Brush, 68
 - Circle, 35
 - Circuit type, 21
 - Clear, 37
 - Close, 36
 - Combined Element, 35
 - Coordinates, 24
 - Del, 68
 - Displacement, 43
 - Draw-Get-LibItem, 40, 68, 97
 - Draw-I/OPorts, 29, 44
 - Draw-SPlanes/boundaries, 26, 38, 49
 - Edit, 35
 - Edit Element, 30
 - Edit Line, 29
 - Edit Object, 30
 - Edit Point, 28
 - Edit Port, 28
 - Edit-Element, 32
 - Edit-Join, 32

- Edit-Object, 32
- Edit-Point, 28, 29, 35, 53
- Edit-Port, 47
- Edit-Redo, 34
- Edit-Reproduce, 30, 32
- Edit-Select Element, 28, 35, 44
- Edit-Select Object, 30
- Edit-Undo, 34
- Element, 34
- Elements, 43
- Exit, 8
- Export, 7
- Export Options, 7
- Export&Run, 7
- Export&Test, 7
- Export,Run&Start, 7
- Extents, 37
- File-Exit, 17
- File-Export, 17, 69, 70, 105
- File-Export Options, 17
- File-Export&Run, 17, 70, 105
- File-Export&Test, 17
- File-Export,Run&Start, 17
- File-Export,Run&Start Prony, 17
- File-Load, 17
- File-New, 17
- File-Save, 17, 69
- File-Save as, 17, 37
- File-UDOPath, 17
- Get, 36, 88
- Grid/Snap, 24
- Help-About..., 17
- Help-Contents..., 17
- Help-Info, 38
- Info-Mesh, 26, 38
- Insert, 68
- IOPorts, 36
- Join, 34
- Level, 23
- Load, 68
- Mesh, 25, 38
- Modify, 31
- New, 68
- Next, 36, 68
- NTF box, 36
- Object, 36
- Open, 36
- Options, 37
- Parameters-Amigo, 17
- Parameters-Circuit type, 17, 21
- Parameters-I/O Ports, 17, 48
- Parameters-Media, 17, 63
- Parameters-Units, 17
- Pen, 68
- Plane, 22, 23
- Plane wave box, 36
- Project- Draft Modify All, 17
- Project- Final Modify All, 17
- Put, 88
- Rectangle, 34
- Redo, 28, 34
- Refresh, 13
- Reproduce, 33
- Save, 43, 68
- Select Element, 31, 32, 36, 48
- Select Object, 32
- Set Plane, 48, 50
- Setup-Axes, 25
- Setup-Level, 28, 34, 36
- Setup-Mesh, 38, 72, 96
- Setup-Plane, 18
- Shift, 48, 50
- SPlanes/boundaries, 36
- Switch, 43
- Tools-ACIS Viewer, 17
- Tools-Calculator, 17
- Tools-DXF Converter, 17
- Tools-SAT Filter, 17
- Tools-SAT Viewer, 44
- Tools-Simulator, 17
- Tools-UDO Editor, 17
- Undo, 27, 34, 104
- Windows-Arrange, 17
- Windows-Open- ProjectInfo, 17
- Windows-Open-2D, 17
- Windows-Open-3D, 17
- Windows-Refresh, 17
- Wire, 35
- Zoom, 37, 43
- Courant stability criterion, 38
- dialogue windows
 - 2D view options, 37
 - 2D-Axes Setting, 24
 - AMIGO, 10
 - Circuit, 21
 - Coordinates Options, 25
 - Define Level, 23
 - Define Plane, 22
 - Draw-Object, 84
 - Edit I/O port, 29
 - Edit special planes, 48
 - Edit special, boundary, 50
 - Element change, 32
 - Element Change, 30
 - Export option, 8
 - Grid/Snap Parameters, 24
 - I/O ports, 45
 - I/O Ports Parameters, 54, 56, 57, 58, 61
 - Join, 34
 - Keyboard Entry, 28
 - Mesh /Splanes info, 26
 - Object Change, 31
 - Parameters-Media, 63, 65, 66, 67, 68, 79
 - Plane shift/rotation, 23
 - Processing/Postprocessing, 59, 61
 - Reproduce, 34
 - Select element, 31, 33, 34
 - Select object, 33
 - Setup-Mesh, 25
 - Special planes and boundaries, 49
 - View Elements, 44
- discretization, 5, 53
- dispersion model
 - Debye, 65
 - Drude, 65
 - Lorentz, 65
- Draw group of commands, 34
- dynamic template, 54
- efficiency, 6
- E-field, 60
- E-field components, 5, 41, 60
- examples, 73, 83, 88, 93

- corhorn3.pro, 19
- wgtocx1.pro, 17, 70
- FDTD, 5
- FDTD cell, 25, 38, 45, 47
- FDTD iterations, 62
- FDTD mesh, 22, 25, 45, 47, 49, 50
- FDTD method, 5
- FDTD simulation, 33, 50, 59, 61, 70
- field distribution, 5, 6, 55, 59
- File group of commands, 7
- files
 - *.bmp, 97, 105
 - *.iop, 87, 98, 105
 - *.mlb, 105
 - *.pa3, 70, 105
 - *.pro, 7, 69, 97, 105
 - *.sh3, 69, 70, 71, 105, 106
 - *.ta3, 105
 - *.txt, 72, 98
 - *.udo, 72, 73, 75, 85, 90, 97, 98, 105
- files *.bmp
 - nobitmap.bmp, 74
- files *.env
 - zedglob.env, 105
- files *.exe
 - qed.exe, 104
- files *.ini
 - zednqw.ini, 14, 27, 104
- files *.iop
 - file_name.iop, 87
- files *.mlb
 - default.mlb, 63, 105
 - standard.mlb, 63
 - standard_with_food.mlb, 63
- files *.pa3
 - wgtocx1.pa3, 70
- files *.pr1
 - project_name.pr1, 105
- files *.pr2
 - project_name.pr2, 105
- files *.pro
 - corhorn3.pro, 19
 - emptyex.pro, 94
 - project_name.pro, 105
 - wgtocx1.pro, 17, 70
- files *.sh3
 - wgtocx1.sh3, 70
- files *.ta3
 - wgtocx1.ta3, 70
 - wgtocx10.ta3, 70
- files *.txt
 - points.txt, 82, 83
 - project_name.txt, 105
- files *.udo
 - cxvld.udo, 74, 75
 - portox.udo, 88
 - taper.udo, 83, 84
 - threeel.udo, 76, 77, 82
 - tool.udo, 93, 94
 - verstep.udo, 73
 - verstepa.udo, 73, 74
 - wiregrid.udo, 40
- Finite Difference Time Domain method, 5
- Fourier transform, 5, 46, 59, 60
- Gaussian beam, 57
- Generalised Pencil of Function Method, 62

- hard edges, 12
- hard planes, 12
- Help group of commands, 16
- H-field, 60
- H-field components, 5
- Info and Help groups of commands, 38
- Info group of commands, 38
- installation, 27, 73, 104
- keys
 - E, 28, 30
 - K, 28, 34, 45
- keywords, 105
- library objects, 69
- lumped impedanc, 57
- lumped probes, 56
- lumped sources, 56
- media library, 63, 77, 105
- media types
 - Dielectric anisotropic, 64
 - Dielectric dispersive, 64, 65
 - Dielectric isotropic, 64
 - Ferrite, 64
 - Metalic, 64
 - Metamaterial, 65
 - PEC, 64
- NTF background, 63
- NTF boundary, 36
- NTF box, 36, 61, 86
- NTF frequencies, 61
- optimisation, 73, 98
- optimiser, 98
- options
 - [S]-differential, 60
 - [S]-direct, 61
 - Cut, 34
 - Decade, 22
 - DXF Converter, 14, 15
 - Edges force linear mesh, 26
 - Eigen values, 61
 - FD-Monitors, 61
 - FD-Pavailable, 60
 - FD-probing, 59
 - Glue, 34
 - Intersect, 34
 - My Tools, 14
 - Narrow, 22
 - Near-to-Far, 61
 - Prony method, 62
 - QW MultiSim, 62
 - SAT Filter, 14, 15
 - SAT Viewer, 14, 15
 - Set Tools, 14
 - Shift&Rotate, 33
 - Sk1 at reference planes, 61, 62
 - Smm at reference planes, 62
 - suppress, 22
 - Test Mesh, 72
 - Two Decades, 22
 - UDO Editor, 14
- Parameters group of commands, 8
- PEC, 64
- Perfect Electric Conductor, 64
- PML, 47
- polar coordinates, 24
- power, 5, 6, 52, 81, 103
- power balance, 6

- power radiated, 6
- Project group of commands, 13
- project media, 63, 68
- project media list, 68
- project tasker file, 70
- project units, 35, 41
- projects
 - emptyex.pro, 94
 - wgtocx1.pro, 31, 48, 70
- Prony method, 62
- Q-factor, 6, 46
- QProny, 6, 62
- QW-3D, 5, 55, 63, 73, 104
- QW-BHM, 8, 64
- QW-Editor, 5, 7, 8, 18, 25, 27, 28, 47, 50, 68, 69, 70, 71, 72, 73, 86, 97, 104, 105
- QW-Editor menu, 7
- QW-Editor toolbar, 17
- QW-ObjectGenerator, 13, 73, 90
- QW-Optimiser, 72, 73
- QW-OptimiserPlus, 72, 73, 98
- QW-Simulator, 6, 14, 15, 17, 49, 53, 60, 64, 105
- radiated power, 6
- radiation efficiency, 6
- radiation patterns, 6, 36, 61
- reference plane, 31, 44, 46, 47, 48, 50, 60, 86
- SAR, 22, 64
- Setup group of commands, 21
- shortcuts
 - Ctrl+A, 17
 - Ctrl+S, 17
 - Ctrl+T, 17
 - Ctrl+Y, 39
 - Ctrl+Z, 39
 - E, 28, 30
 - K, 28, 34, 45
- S-matrix, 5, 46, 47, 60, 62
- S-parameters, 6, 9, 47, 48, 52, 61
- Specific Absorption Rate (SAR), 64
- tasker file, 6, 70, 105
- TEM mode, 54
- template generation, 48, 85
- toolbar
 - 2D window, 38
 - QW-Editor, 17
- Tools group of commands, 13
- UDO, 73, 85, 88, 90, 96, 99
- UDO commands, 77, 99
 - ADDLINE, 77, 79, 81
 - ADDX, 77, 79
 - ADDXYZ, 79
 - ADDY, 77, 79
 - ADJUSTP, 80
 - ALLOWBHM, 90
 - ALLOWHF, 90
 - ANGVAR, 90
 - CALL, 74, 75, 97
 - CIRTYPE, 90
 - CLOSELINE, 77, 79
 - CLOSEOBJ, 74, 76
 - CODE, 82
 - COPY, 92
 - CREATEMESH, 96
 - DELETE, 92
 - DIAMETER, 79
 - DISPERSION, 79
 - ELEMENT, 77, 78
 - ENDELEM, 77, 78
 - ENDHEADER, 74
 - ENDPORT, 86, 88
 - ENDSECTION, 80
 - EPSEFF, 87
 - EVS, 90
 - EXPANDMESH, 95
 - FDPROB, 89
 - GETIOPAR, 86, 87, 88, 98
 - GETMESHPAR, 96
 - GRIDHV, 80
 - INSERTMEDIUM, 79
 - JOIN, 91
 - LOSSES, 90
 - LUMPED, 87
 - LUMPIMPPAR, 87
 - MARK, 91, 92, 93
 - MARKFJ, 91
 - MEDIUMCOL, 79
 - MEDIUMPAR, 77, 79
 - MESH, 95, 96
 - MESHINDEX, 95, 96
 - MESHPAR, 96
 - MESHX, 96
 - MESHXINDEX, 96
 - MESHY, 96
 - MESHYINDEX, 96
 - MIRROR, 92
 - MIRRORX, 92
 - MIRRORZ, 92
 - MODE, 88
 - MTEM, 88
 - NEWLINE, 77, 79, 81, 85
 - NTF, 89
 - NTFBKG, 89
 - OPENF, 81, 82, 98
 - OPENOBJECT, 74, 76
 - PAR, 75, 98
 - PERIODIC, 90
 - PLWAVE, 88
 - PLWBEAM, 88
 - PORT, 86
 - PORTEXC, 85, 86, 87
 - PORTPAR, 86, 87, 95, 96
 - PORTPAR2, 86, 87
 - PORTPOSTP, 86, 87
 - PROFILE, 88
 - PRONY, 90
 - READF, 81, 82, 83, 98
 - READS, 82
 - REDRAW, 92
 - RENAME, 92
 - ROTATE, 92
 - SECTION, 80
 - SETATTR, 78
 - SETOBJMED, 79
 - SETPEN, 93
 - SETSUSPFLAGS, 95
 - SHIFTM, 92
 - SK1DIFF, 89
 - SLICINGPHASE, 95
 - SMNDIFF, 89
 - STL, 82
 - STR, 82

- SYMMETRY, 88
- TEMPD, 88
- TEST, 74, 75
- THERMALPAR, 79
- UNITS, 90
- VAR, 80, 82
- VAWEFORM, 88
- VAWEFORM2, 88
- XCOORD, 80
- YCOORD, 80
- ZCOORD, 80
- UDO interpreter, 75, 81, 96
- UDO language, 70, 73, 80, 83, 85, 91, 96
- User Defined Libraries, 69
- View group of commands, 37, 42
- windows
 - Info-Mesh, 27
 - Level, 23
 - Line change, 29
 - Point Change, 28
- Windows group of commands, 12